Service-Integration in
Geschäftsprozessmanagementsystemen
mit besonderem Fokus auf
die Integration von ERP-Systemen
unter Berücksichtigung des aktuellen
Trends hin zum Cloud-Computing

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften

der Technischen Universität Dortmund

an der Fakultät für Informatik

von
Markus Doedt

Dortmund
2013

Tag der mündlichen Prüfung:	12.12.2013
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter:	Prof. Dr. Bernhard Steffen
	Prof. Dr. Jakob Rehof

Danksagung

„To succeed in business it is necessary to make others see things as you see them.“

John H. Patterson (1844 - 1922)

Ich möchte mich ganz herzlich bei Prof. Dr. Bernhard Steffen bedanken. Er hat als mein Chef und Doktorvater während meiner ganzen Zeit am Lehrstuhl für Programmiersysteme immer ein offenes Ohr gehabt und meine Arbeit stets mit kompetenter und konstruktiver Kritik sowie mit inspirierenden Ideen stark nach vorne gebracht. Ich habe ihn in der Zeit nicht nur als kompetenten Doktorvater und Informatiker sondern auch als sehr lieben Menschen kennen gelernt. Mein weiter Dank geht an meinen Zweitgutachter Prof. Dr. Jakob Rehof, den ich ebenso sowohl als Informatiker als auch als Mensch schätze.

Ganz besonders bedanke ich mich auch bei meiner Familie. Meine Frau Karin, mein Sohn Titus und zuletzt auch meine kleine Tochter Sophie haben oft auf mich verzichten müssen, wenn ich Freizeit für die Erstellung dieser Arbeit geopfert habe. Ich möchte mich ganz herzlich für die viele Geduld und das Verständnis bedanken.

Ich möchte mich auch bei allen bedanken, die während meiner Zeit am Lehrstuhl direkt am Entstehen dieser Arbeit mitgewirkt haben. Dies sind zum einen die vielen Diplomanden mit ihren Arbeiten, zum anderen die studentischen Hilfskräfte, die besonders bei Implementierungsaufgaben behilflich waren und auch ganz besonders meine netten Kollegen und Kolleginnen an der TU Dortmund und an der Universität Potsdam, mit denen ich eine sehr schöne und produktive Zeit verbracht habe. Ich bedanke mich hier besonders bei Sven Jörges, der diese Arbeit korrekturegelesen und mir durch seine konstruktive Kritik sehr geholfen hat.

Zusammenfassung

Durch die konsequente Anwendung eines durchdachten Geschäftsprozessmangements kann ein Unternehmen erreichen, sich seiner Abläufe zunächst bewusst zu werden und diese dann stetig zu verbessern. Doch nur wenn diese Geschäftsprozesse auch automatisiert werden, die Prozessdefinitionen also direkt auf entsprechenden Engines zur Ausführung gebracht werden, erreicht man echte Agilität mit einem effizienten Zusammenspiel von Business und IT. Diese Prozessautomatisierungen bestehen aus Orchestrierungen von Services, welche in einer „service-orientierten Architektur“ (SOA) die verschiedensten IT-Systeme kapseln, die wiederum die gesamten Daten des Unternehmens enthalten. Das effiziente Zusammenspiel von Geschäftsprozessmanagement und SOA hängt stark davon ab, wie gut von beiden Seiten auf die andere zugegangen wird. Auf der Seite der Prozesse ist es notwendig, dass die entsprechenden Modellierungstools und Ausführungsengines es erlauben, möglichst einfach, schnell und fehlerunanfällig Services einzubinden. Auf der Seite der Services müssen diese gut strukturiert und dokumentiert sein, damit sie gefunden und korrekt eingebunden werden können. In dieser Arbeit wird analysiert, wie sich die Situation heute an beiden Seiten darstellt. Ein besonderes Augenmerk wird dabei auf das Thema „Cloud-Computing“ gelegt, da gut beschriebene Services in diesem Bereich von existenzieller Wichtigkeit für die Anbieter sind. Weiterhin wird gezeigt, wie das Zusammenspiel mit Hilfe der Ideen aus dem Konzept „Extreme Model Driven Design“ (XMDD) besonders effizient und elegant realisiert werden kann. Einen besonderen Schwerpunkt bildet die Generierung von Prozessaktivitäten für große Servicesammlungen zum Beispiel von ERP-Systemen.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Die typischen Systemlandschaften in heutigen Unternehmen sind meist sehr heterogen. Neben den zentralen ERP-Systemen (Enterprise Resource Planning) gibt es oft noch viele weitere Lösungen, z. B. als selbst geschriebene Software oder sogar auf der Basis von Tabellenkalkulationen. Häufig kommen auch noch sehr alte Systeme (so genannte „Legacy-Systeme“ [BLWG99]) zum Einsatz, welche oft nur schwer zu warten sind, da sie in einer Programmiersprache entworfen wurden, die heute kaum noch jemand beherrscht. Das gutes Beispiel hierfür ist COBOL, welches vorwiegend bei Banken und Versicherungen aber auch bei großen Industrieunternehmen zum Einsatz kommt. Die Einführung von ERP-Systemen sollte bereits dafür sorgen, dass im Unternehmen ein globales „Integriertes Informationssystem“ existiert, also alle Daten an einem Ort vorgehalten werden. Vorher war die Situation noch problematischer: Für jeden Einzelzweck wurde entsprechende Individualsoftware entwickelt, also z. B. ein System für das Lager, eines für die Produktion und eines für das Marketing. Dies kann schnell zu Dateninkonsistenzen führen, meist einhergehend mit einer unnötigen Dopplung von Arbeit. Die Einführung von ERP-Systemen als „All-in-One-Lösung“ brachte nicht ganz den erhofften Erfolg. Nach wie vor existieren nicht alle Daten in diesem System. Vieles wird zum Beispiel gerne in Tabellenkalkulationen verwaltet, besonders wenn Lösungen in sehr kurzer Zeit entwickelt werden sollen und wenn Personen involviert sind, die selbst keine IT-Fachleute sind. Außerdem hat man mit einem ERP-System nun einen monolithischen Block von genau einem Hersteller, der den Anspruch erhebt, alles in sich zu vereinen, auch das gesamte Geschäftswissen in Form von Ge-

schäftsprozessen. Somit ist man mit einem ERP-System sehr festgelegt auf den jeweiligen Hersteller und ein Umstieg auf andere Lösungen wird zunehmend schwieriger [KVD00]. Diese Situation wird meist mit dem Begriff „Vendor Lock-in“ beschrieben. Mit Verschmelzungen und gegenseitigen Übernahmen („Merger and Aquisitions“) kommt es außerdem oft dazu, dass mehrere ERP-Systeme in einem Unternehmen existieren. Diese oft sehr unterschiedlichen und somit inkompatiblen Systeme zu integrieren oder durch ein neues System abzulösen stellt grundsätzlich eine große Herausforderung dar. Aber nicht nur der Wechsel von bestimmten ERP-Systemen auf andere kann problematisch sein, selbst eine „einfache“ Migration auf eine neue Version des gleichen ERP-Systems kann die IT-Abteilung mit großen Problemen konfrontieren. Das liegt meist daran, dass das ERP-System nicht in seinem nackten Auslieferungszustand als Standard-Software eingesetzt wird, sondern „customized“, also durch eigene Erweiterungen an die Bedürfnisse des Unternehmens angepasst [Hes09]. Möchte man nun das ERP-System auf die neue Version bringen, ist es manchmal nötig, das Customizing für die neue Version erneut durchzuführen, da es so sehr in die alte Lösung „vergraben“ wurde, dass es auf die neue Version nicht anwendbar ist.

Um solche komplexen und heterogenen Systemlandschaften zu integrieren und dieser irgendwie „Herr zu werden“ bietet es sich an, eine Serviceorientierte Architektur (SOA) [Erl05] aufzubauen. Hier werden die Funktionalitäten und Daten der einzelnen Systeme durch so genannte Services gekapselt und nach außen einheitlich angeboten. Diese Services können nun wiederum zu größeren Services kombiniert werden und letztlich maßgeschneiderte, zusammengesetzte Applikationen („custom composite applications“) [BZKP09] bilden. Diese können so entwickelt werden, dass sie die speziellen Anforderungen des Unternehmens möglichst gut erfüllen und gleichzeitig sehr schnell an neue Gegebenheiten angepasst werden können. Gerade in einer schnelllebigen Zeit wie heute ist diese Agilität von besonderer Wichtigkeit. Gleichzeitig kann so erreicht werden, dass das Customizing außerhalb der Standard-Software stattfindet und Migrationen zu neuen Versionen oder ganz neuen Systemen erleichtert werden.

Die Kombination einzelner Services zu größeren wird oft auch „Orchestrierung“[1] genannt. Technisch umgesetzt wird die Orchestrierung durch eine Prozessbeschreibung, die definiert, wie sich der große Prozess aus den einzelnen kleinen Prozessen zusammensetzt. Das bekannteste Beispiel für eine Sprache, mit der solche Orchestrierungsprozesse beschrieben werden, ist WS-BPEL („Web Services - Business Process Execution

[1] Der Begriff „Orchestrierung“ ergibt sich aus der Tatsache, dass hier die Services mit den Musikern eines Orchesters verglichen werden und jeweils eine zentrale Instanz existiert (der Dirigent bzw. eine zentrale Steuerungseinheit), welche festlegt, wie alles zusammenspielt.

Language“) [OAS07]. Der Name dieser Sprache sagt schon viel darüber aus, welche Aspekte hier eine Rolle spielen. Als Technologie für die Services werden hier „Web-Services“ eingesetzt, die wohl immer noch am weitest verbreitete Technologie für Services. Weiter deutet der Name darauf hin, dass es hier auch um „Business Processes“, also um Geschäftsprozesse, geht. Wie diese mit der Orchestrierung von Web-Services zusammenhängen, wird vielleicht nicht auf den ersten Blick klar. Bei genauerer Betrachtung erkennt man, dass die Orchestrierungsprozesse genau den Prozessen in einem Unternehmen (also den „Geschäftsprozessen“) entsprechen. Der letzte Teil des Namens der Sprache besagt „Execution Language“, also „Ausführungssprache“. Dass man Geschäftsprozesse wirklich ausführt war in der Geschichte des Geschäftsprozess-Managements (BPM - Business Prozess Management) [HAHW03] nicht immer so. Anfangs wurden hier eher Betriebsabläufe in Flussdiagramm-ähnlichen Strukturen dokumentiert. Ein bekanntes Beispiel dafür sind die Ereignisgesteuerten Prozessketten (EPKs) [STA05] des ARIS-Konzepts (Architektur integrierter Informationssysteme) [SS06] von August Wilhelm Scheer. Durch die Ausführbarkeit der Geschäftsprozesse hat man nun den Vorteil, dass Modell und Wirklichkeit nicht einfach auseinander driften können (da die Modellierung gleichzeitig Teil der Implementierung ist) und man gleichzeitig ein erhöhtes Maß an Agilität erreicht.

Durch die Einführung von SOA und BPM ist es nun auch möglich, den (ausführbaren) Geschäftsprozess als oberste Instanz anzusehen, der über allen anderen Systemen (also auch dem ERP-System) liegt. So ist der Prozess nicht mehr Teil des Customizings im ERP-System, sondern oberhalb davon angesiedelt. Der Prozess bleibt allein geistiges Eigentum des Unternehmens ohne zum Hersteller des ERP-Systems zu wandern. Das ERP-System wird nun zunehmend als reines Informationssystem behandelt.

Das Zusammenspiel von BPM und SOA ist somit ein Schlüssel zu einer beherrschbaren Systemlandschaft in Unternehmen [Beh06]. Dieses Zusammenspiel wird besonders von Beratungshäusern viel beworben und hoch gelobt. Wenn man nun hinter die Fassaden guckt, merkt man, dass genau an der Schnittstelle zwischen beiden Konzepten die Technologie noch sehr wenig ausgereift ist und dass hier noch so gut wie keine Standards existieren, die definieren, wie genau Services so in ausführbare Geschäftsprozesse integriert werden können, dass dies möglichst fehlerfrei und mit vertretbarem Aufwand möglich ist. Gerade das gute Zusammenspiel von BPM und SOA stellt gleichzeitig einen großen Schritt in Richtung eines guten „Business-IT-Alignments“ dar, also der Zusammenarbeit der unterschiedlichen Stakeholder in einem Unternehmen: auf der einen Seite das Management - auf der anderen die IT-Abteilung.

1.1 Thema der Arbeit

Diese Arbeit beschäftigt sich mit der Problematik der „Service-Integration“ in Geschäftsprozess-Management-Systemen (BPMS - Business Process Management Systems). Dies wird je nach verwendeter Sprache und nach verwendeter Software sehr unterschiedlich umgesetzt und dabei oft noch sehr unstrukturiert. Besonderer Fokus wird hier auf die Ansteuerung von Diensten gelegt, die zu den großen Dienstesammlungen von ERP-Systemen gehören. Dazu wird eine Reihe verschiedener, repräsentativer APIs von ERP-Systemen systematisch untersucht, bewertet und die dahinter liegenden Konzepte analysiert und kategorisiert. Da sich herausgestellt hat, dass aktuelle Entwicklungen im Bereich des „Cloud-Computing“ sich ganz besonders auf die Natur der Services ausgewirkt haben, wird auch dieser Aspekt mit in die Untersuchungen einbezogen. Es wird gezeigt, dass der XMDD-Ansatz[2] bzgl. der Service-Integration das Geschäftsprozessmanagement deutlich vereinfachen kann. Durch die Implementierung eines Frameworks zur Generierung von Prozessbausteinen, die im Prozessmodellierungstool jABC verwendet werden können, wird gezeigt, dass es möglich ist, dass auch Prozessmodellierer ohne technisches Know-How ausführbare Geschäftsprozesse erstellen und zum Einsatz bringen können.

1.2 Thesen und Anforderungen

Im Folgenden werden 5 Thesen (bezeichnet mit T1 bis T5) aufgestellt, aus denen sich die Anforderungen dieser Arbeit direkt ergeben. Die Anforderungen bestehen konkret darin, diese Thesen durch Untersuchungen und eigene Implementierungen zu belegen oder bei Bedarf entsprechende Einschränkungen aufzuzeigen.

T1 - Enttäuschende Qualität von ERP-APIs Die Schnittstellen von ERP-Systemen haben im Allgemeinen eine sehr enttäuschende Qualität. Außerdem gibt es keinen Standard für die Entwicklung von ERP-APIs. Daraus folgt, dass die Umsetzungen jeweils extrem unterschiedlich aussehen können. Selbst die Benutzung derselben grundlegenden Technologie (z. B. WSDL und SOAP) ändert daran nichts.

T2 - Cloud-Computing führt zu besseren APIs Der Trend, Systeme (z. B. ERP-Systeme) in die Cloud zu verlagern (oft als SaaS) übt einen enormen Druck auf die Hersteller aus, ihre APIs qualitativ zu verbessern.

[2]XMDD: Extreme Model Driven Design, siehe Abschnitt 2.6.1

Das führt dazu, dass die APIs der Cloud-Systeme viele Schwachpunkte der APIs traditioneller Systeme nicht mehr aufweisen und teilweise sogar überraschend gute und kreative Lösungen bieten.

T3 - Aktuelle BPMS unterstützen wenig bei der Service-Integration Aktuelle Geschäftsprozessmanagementsysteme konzentrieren sich meist stark darauf, wie Prozesse modelliert werden, wie Benutzerinteraktion und Aufgabenverwaltung realisiert wird und wie die einzelnen Elemente der Sprache unterstützt werden. Der Punkt der Service-Integration wird meist nur sehr stiefmütterlich behandelt. Das führt vor allem zu einer hohen Fehleranfälligkeit. Oft müssen z. B. bestimmte Bezeichner, die an unterschiedlichen Stellen eingegeben werden, genau übereinstimmen, ohne dass das Modellierungstool überprüft ob dies der Fall ist.

Das Vorgehen zur Integration eines Services in den Prozess verlangt bei aktuellen BPMS ein erhebliches Maß an technischem Verständnis. Das liegt vor allem daran, dass die jeweils verwendete Technologie im Mittelpunkt steht, mit der der Service angeboten wird. Mit dieser Technologie und den damit zusammenhängenden Begrifflichkeiten muss sich der Anwender genau auskennen um den Service zu integrieren. Daraus folgt, dass ein typischer Anwendungsexperte ohne technisches Know-How nicht in der Lage ist, den Prozess so zu modifizieren, dass ein neuer Service eingebunden wird oder dass eine bestimmte Service-Integration abgeändert wird.

Der manuelle Aufwand, einen bestimmten Service in einen Prozess zu integrieren, ist bei aktuellen BPMS meist extrem aufwändig. Dies liegt oft daran, dass bestimmte Handlungen mehrmals durchgeführt werden müssen, obwohl dies eigentlich nicht nötig wäre.

T4 - XMDD und dessen SIB-Konzept können Service-Integration vereinfachen Das den SIBs[3] zugrundeliegende Konzept ist eine fortschrittliche Art und Weise, wie Services eingebunden werden können. Dieses Konzept behebt viele der oben genannten Schwachstellen anderer BPMS in Bezug auf das Service-Integration-Thema.

T5 - Eine Wizard-gesteuerte Generierung von SIBs vereinfacht Service-Integration weiter, besonders bei großen APIs Für die APIs von ERP-Systemen ist eine extreme Größe und Komplexität typisch. Man hat hier eine große Menge von relativ gleichförmig aufgebauten Services. Gerade für diesen Zweck eignet sich der Einsatz von Codegenerierungstechniken. Der Einsatz eines Wizards und die damit verbundene

[3]SIB: Service Independent Building Block, siehe Abschnitt 2.6.3

Beschränkung der Automatisierung auf eine halbautomatische Lösung bewirken, dass der Benutzer so Einfluss auf die Generierung ausüben kann, dass die Services so aussehen, wie er es sich wünscht. Technische Probleme müssen bei diesem Ansatz meist nur einmal gelöst werden, nämlich bei der Implementierung der Wizards und nicht mehr bei jeder Service-Integration neu.

1.3 Meine Arbeitsschwerpunkte

Der Kern der geleisteten Arbeiten, die dieser Ausarbeitung zu Grunde liegen, gliedert sich grob in drei Teile:

1. Untersuchung von ERP-APIs
2. Untersuchung von BPMS bzgl. Service-Integration
3. Entwicklung eines Service-Integration-Frameworks für ERP-Systeme durch Wizard-gesteuerte Generierung von Prozessbausteinen

Untersuchung von ERP-APIs: In einer ersten Studie (siehe Abschnitt 3.2) wurden vier verschiedene APIs von traditionellen ERP-Systemen untersucht. Dazu gehören zwei Lösungen des Marktführers SAP und zwar die ältere BAPI/RFC-Lösung sowie die modernere, Webservice-basierte eSOA-Lösung. Als Kontrast wurde neben dieser Lösung für sehr große Unternehmen das deutliche schlankere System „Dynamics NAV" von Microsoft gestellt, welches eher auf kleinere und mittlere Unternehmen abzielt. Als weiteres System wurde noch Intuit Quickbooks in die Untersuchung mit aufgenommen, ein Buchhaltungssystem, was auf Grund seines Umfangs aber auch schon als ERP-System bezeichnet werden kann. Dieses System ist besonders in den USA sehr erfolgreich und zielt eher auf kleinere Unternehmen ab. Diese Studie zeigte, dass die APIs eine erschreckend geringe Qualität aufwiesen und es wird deutlich, dass der Entwicklung der API von Herstellerseite bei weitem nicht der Stellenwert eingeräumt wird wie der Entwicklung der graphischen Benutzeroberflächen. Ebenso konnte an den mangelnden Features der APIs abgelesen werden, welches Geschäftsmodell die Hersteller verfolgen. Anscheinend bekommt die Bereitstellung einer guten API in vielen Fällen keine hohe Priorität bei den Herstellern. Wenn die nahtlose Integration mit Systemen anderer Hersteller erschwert wird, bedeutet dies automatisch eine stärkere Bindung der Nutzer an den einen einzelnen Hersteller und somit die Förderung einer „IT-Monokultur".

Für die genannte Studie wurde ein Anforderungskatalog entwickelt, der beschreibt, welche Eigenschaften eine gute ERP-API aufweisen sollte,

damit die dahinter liegenden Systeme möglichst gut integrierbar sind. Besonderer Augenmerk wurde darauf gelegt, ob die APIs hinreichend Informationen anbieten, um auf dieser Basis automatisch Prozessbausteine zu generieren, die in BPMS zur Entwicklung ausführbarer Geschäftsprozessbeschreibungen genutzt werden können.

In einer zweiten Studie (siehe Abschnitt 3.3) wurde die gerade genannte Studie um Cloud-ERP-Systeme (also genauer SaaS-Lösungen) erweitert. Dazu gehört die CRM-Lösung Salesforce, das ERP-System NetSuite sowie die HCM-Software Workday. Dabei sollte vor allem die Frage beantwortet werden, ob es insgesamt einen wesentlichen Unterschied zwischen den APIs der traditionellen ERP-Systeme und der Cloud-Lösungen zu beobachten gibt. Diese These konnte im Großen und Ganzen belegt werden. Alle untersuchten Cloud-Systeme boten sehr solide Schnittstellen an. Der Anforderungskatalog, der von der ersten Studie übernommen wurde, konnte zu einem deutlich höheren Grad abgedeckt werden.

Ergänzt wurden die Studien noch von weiteren Untersuchungen von APIs im Business-Umfeld. Zunächst wurde dazu OpenSource-ERP-Software daraufhin untersucht, ob hier ebenfalls APIs angeboten werden und wenn ja, wie gut diese sind[4]. Die Untersuchungen bestätigen besonders die schon in den vorherigen Studien gewonnenen Erkenntnisse, dass die Lösungen einen sehr unterschiedlichen Charakter haben können, was meist in der Historie der Software-Produkte begründet ist. Standards werden sehr unterschiedlich eingesetzt, so dass eine einheitliche Behandlung der APIs nicht möglich ist. Auf jede API muss bei der Integration individuell eingegangen werden, damit der Anwender nicht immer wieder mit den technischen Eigenarten der API-Umsetzung konfrontiert wird. Zusätzlich wurden noch die APIs der E-Commerce-Plattformen Amazon und Ebay betrachtet. Diese sind von der Größe her vergleichbar mit den ERP-APIs und haben durch ihre sehr starke Verbreitung eine besonders große Relevanz. Auch hier zeigt sich wieder, dass es immer noch wieder andere technologische und konzeptionelle Ansätze gibt, eine API umzusetzen. Und auch, wenn beide betrachtete APIs prinzipiell sehr ähnliche fachliche Funktionen anbieten[5], so sind die Umsetzungen doch extrem unterschiedlich.

Untersuchung von BPMS bzgl. Service-Integration: Zu den verschiedenen Konzepten der Service-Integration in Geschäftsprozessmanagementsystemen wurde ebenfalls eine Studie durchgeführt (siehe Kapitel

[4]Diese Untersuchungen wurden in Zusammenarbeit mit der studentischen Hilfskraft Dominic Wirkner durchgeführt.

[5]E-Commerce: Man kann kaufen, verkaufen und andere damit zusammenhängende Handlungen vollziehen.

6). Die Konzepte wurden kategorisiert, bewertet und untersucht, inwieweit die einzelnen Systeme diese Konzepte unterstützen. Dazu wurde jBPM von JBoss in seinen Versionen 4 und 5 untersucht, außerdem Activiti, Bonita Open Solution, AristaFlow, YAWL, Windows Workflow Foundation und BPEL. Diesen Lösungen die XMDD-basierte Lösung des jABC gegenübergestellt (siehe Abschnitt 5.1). Es stellte sich heraus, dass die meisten BPMS nur eine sehr rudimentäre Unterstützung für Service-Integration bieten, was oft zu einer hohen Fehleranfälligkeit führte. Auch war der Integrationsvorgang meist von extrem technischer Natur, so dass der Prozessmodellierer selbst nicht in der Lage war, die Services anzusprechen. Lediglich das jABC und die in dieser Arbeit vorgestellten Konzepte zur halbautomatischen Generierung von Prozesskomponenten ermöglichen es, die Zusammenarbeit zwischen Business und IT so zu organisieren, dass die IT mit geringem Aufwand einmal eine ganze API in Form eines Wizard-gesteuerten Komponentengenerators integriert und anschließend die Businessseite ohne technisches Know-How direkt ausführbare Prozesse modelliert.

Entwicklung eines Service-Integration-Frameworks für ERP-Systeme durch Wizard-gesteuerte Generierung von Prozessbausteinen: Um die Erstellung von Prozessbausteinen (in jABC so genannten „SIBs“) zu vereinfachen, wurde ein Framework mit dem Titel „InBuS“[6] entwickelt, welches dem Benutzer erlaubt, ohne technisches Know-How Services von ERP-Systemen in die Prozesse einzubinden (siehe Abschnitt 5.3.5). Dazu wird dem Benutzer ein Wizard angeboten, über den er die Generierung der SIBs beeinflussen kann, die wiederum die ERP-Services kapseln. So müssen die technischen Probleme der Service-Integration nur noch einmal - und zwar bei der Erstellung der Generatoren - gelöst werden und nicht immer wieder bei jeder einzelnen Service-Integration. Die Wizards werden so entworfen, dass sie trotz unterschiedlicher zu Grunde liegender APIs sehr einheitlich gestaltet sind, so dass der Benutzer damit möglichst schnell zurecht kommt. Bei der Entwicklung neuer Wizards und Generatoren kann man sich hauptsächlich auf die technische Integration der Services konzentrieren, da die Wizard-GUI sehr gut wiederverwendbar ist. Die Generierung erfolgt mit Hilfe einer Template-Sprache. Da die Templates sehr leicht zu erstellen sind, ist es kaum aufwändiger, ein allgemeines Template für die SIBs einer API zu generieren als ein SIB für einen einzelnen Service zu erstellen. Voraussetzung dafür ist natürlich eine einheitlich und konsistent strukturierte API, womit man wieder bei den API-Untersuchungen aus Kapitel 3 angelangt ist.

[6]Integrating Business Software

KAPITEL 2

Grundlagen

In folgendem Kapitel werden grundlegende Begrifflichkeiten definiert, die zum Verständnis der weiteren Ausführungen nötig sind. Zunächst wird in Abschnitt 2.1 der Bereich des Geschäftsprozessmanagement betrachtet. Dazu wird definiert, was ein Geschäftsprozess ist, darauf basierend, was Geschäftsprozessmanagement bedeutet und schließlich was ein Geschäftsprozessmanagementsystem auszeichnet. In Abschnitt 2.2 wird dann auf den Begriff der „Service-orientierten Architektur" eingegangen und dazu Eigenschaften, Vorteile sowie Herausforderungen aus diesem Bereich genannt. Eng verwandt damit ist der Begriff der „Application Programming Interfaces" mit dem sich Abschnitt 2.3 befasst. Anschließend wird in Abschnitt 2.4 der Cloud-Computing-Begriff definiert und erläutert wonach in Abschnitt 2.5 ERP-Systeme behandelt werden. Schließlich wird in Abschnitt 2.6 auf das am Lehrstuhl 5 der TU Dortmund entwickelte XMDD-Konzept und dessen Referenzimplementierung jABC eingegangen.

2.1 Geschäftsprozessmanagement

Das „Geschäftsprozessmanagement"[1] beschäftigt sich (wie der Name schon sagt) mit der Verwaltung von Geschäftsprozessen. Um genau zu verstehen, was damit gemeint ist, soll zunächst definiert werden, was genau

[1] oft auch „Business-Process-Management" (BPM)

ein Geschäftsprozess ist und anschließend erläutert werden, was in diesem Zusammenhang der sehr allgemeine Begriff „Management " alles umfasst.

Geschäftsprozessmanagement ist eine interdisziplinäre Disziplin zwischen der Informatik und verschiedenen Anwendungsgebieten, hier vor allem den Wirtschaftswissenschaften. Dieser interdisziplinäre Charakter bewirkt eine Vielzahl unterschiedlicher Sichten auf das gleiche Thema. Je nachdem, welche Community nach einer Definition des Begriffes „Geschäftsprozess" gefragt wird, bekommt man eine andere Antwort. Lindsay et al. [LDL03] haben daher verschiedenste Definitionen des Begriffes nebeneinandergestellt und die verschiedenen Aspekte beleuchtet. Ähnliches hat auch die OMG[2] durchgeführt und beschrieben, was aus ihrer Sicht ein Geschäftsprozess ist [OMG12]. Die wichtigsten Definitionen seien im Folgenden genannt:

Von Bider [Bid00] stammt die folgende Definition aus dem Jahr 2002:

> „Set of partially ordered activities intended to reach a goal"

Die Definition ist also sehr allgemein gehalten und sagt nur aus, dass eine Menge von „Aktivitäten" vorhanden ist, die auf irgendeine Art geordnet ist (es existiert also eine definierte Reihenfolge) und die Aktivitätenfolge hat insgesamt ein bestimmtes Ziel. Hier wird weder von Informationstechnologie noch von einer Anwendung in einem Unternehmen gesprochen.

Jacobson [JEJ94] definierte bereits 1995 den Begriff folgendermaßen:

> „Set of internal activities performed to serve a customer"

Hier kommt der Aspekt hinzu, dass ein bestimmter „Kunde" („customer") existiert, also irgendjemand, der Interesse an dem Ausgang des Prozesses hat.

Ähnlich definieren Rummler und Branche [RB95] den Begriff im gleichen Jahr:

> „The series of steps that a business executes to produce a product or service"

Hier wird anstelle des Kunden das Ergebnis des Prozesses betont („product or service"). Außerdem wird hier explizit das Unternehmen („business") erwähnt, in dem der Prozess ausgeführt wird.

Gulla and Lindland stellten bereits 1994 fest, dass es eine Verschiebung in der Bedeutung des Begriffes gibt, weg von Produktionsprozessen hin zu Koordinationsprozessen, also Prozessen im Büro bzw. allgemein in

[2]OMG: Object Management Group, http://www.omg.org/

der Verwaltung. Wenn heute von „Geschäftsprozessen“ gesprochen wird, sind also immer Prozesse in der Verwaltung eines Unternehmens gemeint, nicht die eher kleinschrittigen Prozesse zur Produktion eines bestimmten Produktes. Dies ist auch in dieser Arbeit so zu verstehen.

Um nun zu beschreiben, was genau mit „Management“ im Umfeld von Geschäftsprozessen gemeint ist, muss man zunächst in der Geschichte etwas zurückgehen. In den 90er Jahren war ein häufig benutzter Begriff im Zusammenhang mit Geschäftsprozessen „Business Process Reengineering“ [JMPW93]. Damit ist die Tätigkeit der Neugestaltung der Prozesse in einem Unternehmen gemeint, basiert auf intensiven Untersuchungen und Analysen. Insgesamt beschränkte man sich darauf, die Prozesse zu analysieren und zu dokumentieren (also zu modellieren). Die modellierten Ist-Prozesse können dann immer wieder als Grundlage für die Optimierung der Prozesse benutzt werden. Diese Prozesse wurden noch nicht direkt ausgeführt sondern waren maximal Vorlagen für die manuelle Softwareentwicklung ähnlich wie es traditionell meist im UML-Umfeld gemacht wird. Ein bekanntes und bedeutendes Beispiel aus dieser Zeit sind die „Ereignisgesteuerten Prozessketten“ (EPK) [STA05] aus dem ARIS-Konzept[3] [SS06] von August-Wilhelm Scheer. Nach wie vor ist diese Modellierungstechnik weit verbreitet und hat auch aktuelle Standards wie BPMN[4] stark beeinflusst. Zu dieser Zeit stand das Kürzel „BPM“ noch meist für „Business Process Modeling“.

Mit dem Aufkommen service-orientierter Architekturen (SOA, siehe Abschnitt 2.2) wurden die Prozesse zunehmend als „Orchestrierung“ technischer Services gesehen. Das bedeutet, dass nun die Prozesse direkt ausgeführt werden sollten und zwar durch eine so genannte „Process Engine“, oft auch „Process Server“ oder „Execution Engine“ genannt. Durch das Hinzukommen der direkten Ausführung ergaben sich auch direkt weitere Vorteile, wie z. B. die Möglichkeit des „Monitoring“, also der Beobachtung der Prozesse während ihrer Ausführung. Die Beobachtungsergebnisse können nun wieder direkt als Grundlage für die Optimierung der Prozesse benutzt werden. Insgesamt ergibt sich so ein Zyklus bestehend aus Modellierung, Implementierung[5], Ausführung, Monitoring, Optimierung, wieder Modellierung usw. Dies ist der so genannte „Business Process Management Life Cycle“, also ein Kreislauf, dargestellt in Abbildung 2.1. BPM steht nun also nun für den umfassenderen Begriff „Business Process Management“, also Geschäftsprozessmanagement, was

[3] ARIS: „Architektur integrierter Informationssysteme“

[4] BPMN: Business Process Model and Notation

[5] Unter „Implementierung“ fallen hier alle Schritte, die evtl. noch notwendig sind, um den modellierten Prozess ausführbar zu machen. Oft müssen noch technische Details hinzugefügt werden, um diesen Zustand zu erreichen. Im Optimalfall beschränkt sich dieser Aufwand auf das Deployment auf die Engine.

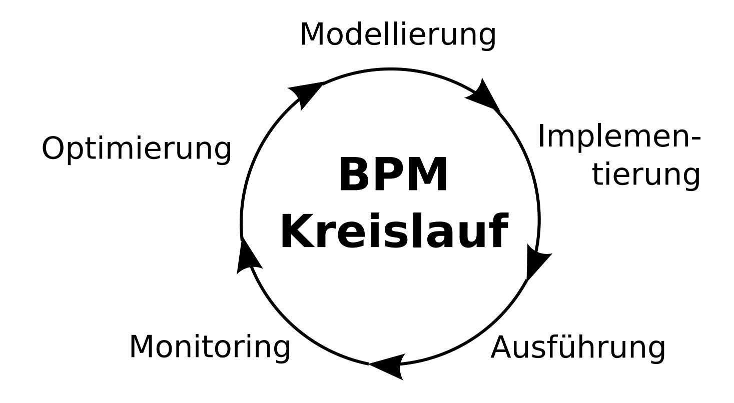

Abbildung 2.1: BPM-Kreislauf

alle Bestandteile des genannten Kreislaufs umfasst.

Das bekannteste Beispiel für eine direkt ausführbare Geschäftsprozesssprache ist WS-BPEL[6], oft auch einfach BPEL genannt. Man merkt hier deutlich, dass die Sprache eher aus der SOA-Welt als aus der BPM-Welt stammt, da hier ein Prozess wirklich nur aus einer Orchestrierung von Services besteht. Prozesse laufen also vollautomatisch ab und jeder Prozessschritt ist der Aufruf eines Web-Services. Echte Geschäftsprozesse, in denen es neben IT-Systemen auch um Menschen geht, die involviert werden müssen, sind zunächst einmal nicht möglich. Erst später wurde daher die Erweiterung BPEL4People (also BPEL für Menschen) eingeführt, was lange Zeit nur ein White Paper ohne Implementierung war und irgendwann (wenn auch oft sehr unterschiedlich) nach und nach von den Engine-Herstellern umgesetzt wurde. In dieser Arbeit werden ausschließlich ausführbare Prozessbeschreibungen betrachtet. Reine Modellierungen zu Dokumentationszwecken spielen hier keine Rolle.

Automatisierte Prozesse wie in BPEL sind auch typisch für den Bereich der „Enterprise Application Integration“ (EAI), wo es darum geht, verschiedenste Applikationen in einem Unternehmen miteinander zu verbinden. Dazu wird traditionell eine nachrichtenbasierte Middleware (MOM - Message Oriented Middleware) eingesetzt. Durch SOA wurde hier die ehemals monolithische EAI-Software durch einen ESB (Enterprise Service Bus) ersetzt, welcher sehr modular aufgebaut ist und verschiedenste Aufgaben in der Kommunikation zwischen den einzelnen Applikationen übernimmt. Neben dem reinen Datenaustausch über Nachrichten kommen hier noch Funktionen wie Message-Routing (Wohin genau muss eine Nachricht übertragen werden?), Datentransformation (z. B. per XSLT), Service-Orchestrierung (z. B. über BPEL-Prozesse) oder Service-Aufrufe über Adapter (z. B. zum Aufruf der konkreten Endsysteme wie SAP-

[6]WS-BPEL: „Web services - Business Process Execution Language“

ERP, einer SQL-Datenbank etc.) hinzu. Somit kann SOA heute als eine Weiterentwicklung von EAI gesehen werden, wobei (Geschäfts-)Prozesse eine besonders große Rolle spielen.

Ein anderer Bereich, der ebenfalls in den 90er-Jahren sehr stark wurde, ist das Workflow Management. Unter „Workflows" werden hier meist Prozesse verstanden, die die Arbeitsabläufe von Personen beschreiben[7]. Sehr oft stehen hier Dokumente im Mittelpunkt, so dass es darum geht, innerhalb eines DMS (Dokumenten Management System) die Prozesse zu steuern, wie Mitarbeiter mit den Dokumenten umgehen, wann also welcher Arbeitsschritt von wem mit einem bestimmten Dokument erledigt werden muss. Die 1993 gegründete WfMC („Workflow Management Coalition")[8] beschäftigt sich mit genau diesem Thema. Sie definierte 1995 das so genannte „Workflow Reference Model" [H+95] (siehe Abbildung 2.2), eine Standard-System-Architektur für Workflow-Management-Systeme. Diese ist so auch heute immer noch gültig für alle Geschäftsprozessmanagementsysteme. Da in dieser Arbeit ausschließlich ausführbare Geschäftsprozesse betrachtet werden, können hier demnach die Begriffe „Workflow" und „Geschäftsprozess" synonym verwendet werden. Man hat damit also neben der dynamischen Sicht des BPM-Zyklus auch eine statische Sicht auf die beteiligten Systeme und wie diese zusammenhängen. Im Zentrum steht dabei die Engine, welche über verschiedene Interfaces mit anderen Systemen interagiert. Definiert werden die Prozesse in einem entsprechenden Modellierungstool, der Workflow kommuniziert mit den beteiligten Endnutzern über entsprechende Clientsoftware bzw. eine „Worklist"[9], der Workflow kann externe Applikationen aufrufen und mit anderen Engines kommunizieren. Die WfMC hat mit XPDL („XML Process Definition Language") auch ihre eigene Prozessdefinitionssprache definiert, welche jedoch heute keine große Bedeutung mehr hat. Zwischenzeitlich wurde noch versucht, XPDL als Serialisierungsformat für die sehr erfolgreiche Notation BPMN (bis Version 1.2: „Business Process Modeling Notation") zu etablieren. Dies ging jedoch schief, da BPMN mit Version 2.0 ein eigenes Serialisierungsformat bekam.

Die sehr manuellen Workflows unterscheiden sich wesentlich von den automatisch ablaufenden Prozessen aus dem oben beschriebenen EAI-Bereich. Heutzutage wachsen unter dem Dach des Geschäftsprozessmanagements und mit dem Aufkommen von ausführbaren Prozesssprachen und entsprechenden Engines diese Arten sehr zusammen. BPMN 2.0 (seit

[7] Je nach Community wird der Workflow-Begriff auch manchmal anders definiert, z. B. synonym zu „(Geschäfts-)Prozess" oder als technischer, kleinschrittiger Prozess oder als Automatisierung eines Geschäftsprozesses

[8] Die WfMC ist ein Verbund von über 300 Mitgliedern, darunter Tool-Hersteller und Nutzer, Berater und Wissenschaftler.

[9] Eine Liste von noch zu erledigenden Aufgaben.

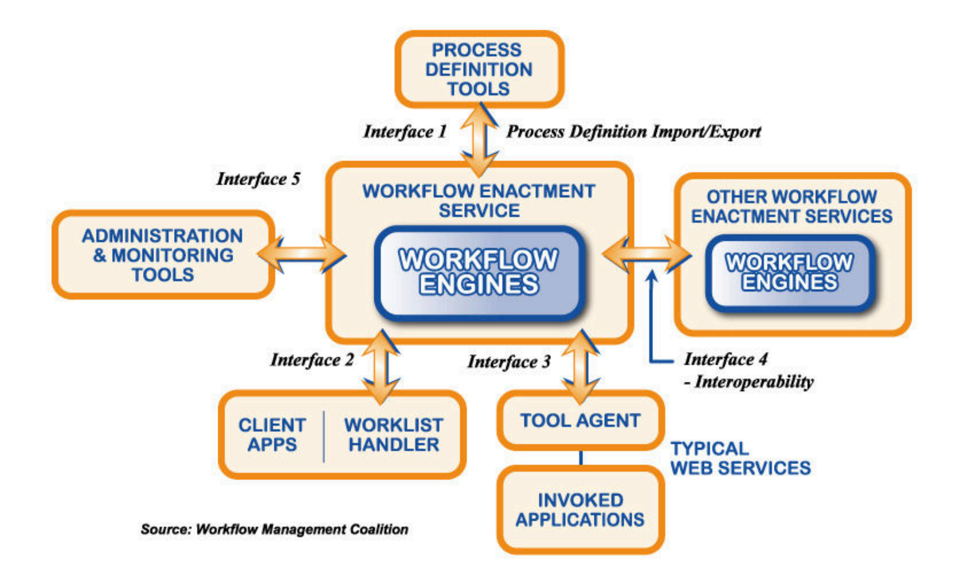

Abbildung 2.2: Workflow Reference Model (Quelle: [WfM95])

dieser Version: „Business Process Model and Notation") ist zum Beispiel eine ausführbare Geschäftsprozesssprache und wird von verschiedenen Engines unterstützt. Momentan sieht es so aus, als entwickle sich BPMN zu dem De-facto-Standard im BPM-Bereich. Da BPMN insgesamt technologieneutral[10] definiert und somit in seiner Spezifikation an einigen (technischen) Stellen recht wage bleibt, ist es den Engine-Herstellern überlassen, wie genau bestimmte Dinge umgesetzt werden. Zum Thema „Service-Integration" wird zum Beispiel nur sehr wenig ausgesagt. In Abschnitt 4.1 wird auf diesen Aspekt genauer eingegangen.

2.2 Service-orientierte Architektur (SOA)

Der Begriff „Service-orientierte Architektur" (kurz „SOA") wird (wie es auch schon bei BPM war) je nach Quelle und Community unterschiedlich definiert. Schon im Jahr 1996 benutzte das Beratungsunternehmen Gartner den Begriff in einem Bericht. Laut einem weiteren Gartner-Bericht von 2003 [Nat03] war dies die erste Nennung und damit Erfindung des Begriffes. Hier wird SOA zusammenfassend folgendermaßen definiert:

[10]BPMN setzt weder eine bestimmte Programmiersprache (wie z. B. Java) noch eine bestimmte Service-Technologie (wie z. B. SOAP-Web-Services) voraus.

> „SOA is a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls."

Es handelt sich also um eine Software-Architektur, wobei hier die Schnittstellen („interfaces") in den Mittelpunkt gestellt werden. Da in dieser Arbeit APIs stark im Fokus stehen, welche im Wesentlichen Sammlungen von Interfaces sind, kann diese Definition hier sehr gut verwendet werden. Gartner merkt auch selbst an, dass der Begriff „interface-oriented Architecture" besser passen würde. Der Begriff „Service" wird hier so definiert:

> „Services are software modules that are accessed by name via an interface, typically in a request-reply mode."

Ein Service besitzt also immer einen Namen, durch den er referenziert wird sowie eine Schnittstelle („interface") und antwortet auf Anfragenachrichten („request") mit Antwortnachrichten („reply"). Genau das ist es, was bei einem Aufruf eines Services von einem Geschäftsprozess aus passiert.

Insgesamt ist die Definition von Gartner noch recht wage gehalten und konzentriert sich auf das wesentliche Merkmal, der Existenz von Services.

Häufig zitiert wird auch eine Definition der OASIS[11] von 2006 [OAS06], die sogar noch vager gehalten ist:

> „A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."

Diese Definition ist sehr technologie-neutral. Betont wird der verteilte Charakter („distributed capabilities") der Ressourcen oder Fähigkeiten („capabilities"), besonders auch die mögliche Verteilung der Zuständigkeiten („different ownership domains"). Weitere genannte Aspekte sind das Anbieten, Auffinden, Interagieren und die Benutzung dieser Ressourcen um erwünschte Effekte zu erzielen mit messbaren Vor- und Nachbedingungen.

Auffällig ist bei beiden Definitionen, dass zunächst einmal nichts von „Web-services" oder sogar einer konkreten Technologie (z. B. WSDL oder

[11] OASIS: „Organization for the Advancement of Structured Information Standards", eine internationale, nicht-gewinnorientierte Organisation, die sich mit der Weiterentwicklung von E-Business- und Webservice-Standards beschäftigt. Der WS-BPEL-Standard wird z. B. von der OASIS gepflegt.

SOAP) gesagt wird. Die Betrachtung der Prozessengines in dieser Arbeit (besonders in den Kapiteln 5 und 6) zeigt auch ganz deutlich, dass sehr oft service-orientierte Architekturen ohne den Einsatz von Web-Services existieren. Dies ist wichtig und wird auch explizit von der deutschen Gesellschaft für Informatik (GI) in ihrem „Informatiklexikon“ [RHS05] so erwähnt:

> „Dabei wird SOA häufig gleichgesetzt mit der Verwendung von Web Services und deren assoziierten Technologien. SOA sollte aber als fachliches Architekturmuster interpretiert werden, das technologieunabhängig angewendet werden kann.“

Definiert wird SOA hier folgendermaßen:

> „SOA ist ein Architekturmuster, das den Aufbau einer Anwendungslandschaft aus einzelnen fachlichen Anwendungsbausteinen beschreibt, die jeweils eine klar umrissene fachliche Aufgabe wahrnehmen.“

Weiterhin werden hier die folgenden Charakteristika erwähnt:

- Es existieren fachliche Anwendungsbausteine mit klar umrissener fachlicher Aufgabe.
- Es besteht eine lose Kopplung zwischen den Anwendungsbausteinen. Funktionalitäten werden als Services angeboten.
- Ein Service ist eine feste, definierte Leistung, kann als Prozesselement verwendet werden, ist eine abstrakte Sicht auf den Anwendungsbaustein, verbirgt Implementationsdetails, hat den Charakter einer vertraglichen Übereinkunft und ist tendenziell grobgranular.
- Der Aufruf von Services geschieht über einen einheitlichen, plattformunabhängigen Mechanismus über anonyme Schnittstellen. Das Auffinden des Service-Anbieters nennt sich Discovery.
- Wichtig ist die Trennung der Zuständigkeiten nach fachlichen Gesichtspunkten („Separation of Concerns“) und die Kapselung technischer Details.

In Punkt 2 („kann als Prozesselement verwendet werden“) wird also deutlich, dass Prozesse in einer SOA eine große Rolle spielen. Somit hängen BPM und SOA stark zusammen. Dieses Zusammenspiel ist genau das, was durch die Betrachtung der Thematik „Service-Integration“ in dieser Arbeit ausgeleuchtet werden soll.

Thomas Erl definiert SOA über acht verschiedene Entwurfsprinzipien [Erl08], wobei es sich dabei um Regeln handelt, die vorgeben, was genau Services sind, wie sie also gestaltet werden sollten[12]:

[12] laut Erl selbst ist ein Entwurfsprinzip ein „in einer Branche generalisierte, allgemein anerkannte Vorgehensweise oder Praxis“[Erl08, S. 41]

1. **Serviceverträge**: Dazu gehört die technische Schnittstellenbeschreibung (im Falle von Web-Services z. B. Dokumente in den Formaten WSDL, XSD und/oder WS-Policy) sowie nicht-technische Vereinbarungen wie SLAs („Service Level Agreements“).
2. **Kopplung von Services**: Zwischen den Services ist eine möglichst „lose Kopplung“ gewünscht. Dies bedeutet eine Minimierung der Abhängigkeiten zwischen den Services und zwischen einem Service und dessen „Consumer“[13].
3. **Abstraktion von Services**: Das Verstecken von Informationen (z. B. der Implementierung), auch „Kapselung“ genannt, ist ein wichtiger Punkt bei Services. Jeder Service ist eine „Blackbox“, nur die Schnittstelle (also die API - siehe Abschnitt 2.3) ist öffentlich.
4. **Wiederverwendbarkeit von Services**: Ein Service ist grundsätzlich wiederverwendbar. Dabei wird nicht nur ein bestimmter Code wiederverwendbar gemacht (wie bei der Objektorientierung), sondern eine konkrete, deployte Instanz einer Softwarekomponente.
5. **Autonomie von Services**: Jeder Service ist ein selbstständiger Baustein.
6. **Zustandslosigkeit von Services**: Die Services selbst haben keinen Zustand, „behalten“ also keine Daten bzw. Informationen.
7. **Auffindbarkeit von Services**: Das Identifizieren des richtigen Services wird auch „Service Discovery“ genannt. Services müssen also in einem Repository vorgehalten werden und dort hinreichend durch Metainformationen beschrieben sein, damit man sie bei Bedarf findet und schließlich benutzen kann.
8. **Kompositionsfähigkeit von Services**: Eine Menge einfacher Services kann durch einen Prozess, der auf diesen Services basiert, zu einem größeren, komplexeren Service kombiniert werden. Dies wird auch oft „Orchestrierung“ genannt.

In Punkt 8 („Kompositionsfähigkeit“) wird deutlich, dass sich Thomas Erl auch auf die Rolle von BPM in SOA [Erl08, S. 114 f.] bezieht:

> „Die Geschäftsprozessebene ist ein Kernstück jeder serviceorientierten Architektur. Aus der Sicht der Komposition hat sie normalerweise die Rolle einer übergeordneten Kontrollinstanz für die Servicekomposition. Das Aufkommen der Orchestrierungstechnologie hat diese Rolle aus der Implementierungsperspektive weiter gestärkt.“

[13] Der „Consumer“ ist die Service-aufrufende Instanz, man könnte also auch „Kunde“ oder „Client“ sagen.

Nicolai Josuttis [Jos08] betont, dass es sich bei SOA nicht um eine konkrete Architektur handelt, sondern eher um eine Art Konzept, welches zu einer Architektur hinführt. Als zentralen Pluspunkt gegenüber vorherigen Konzepten oder Architekturen sieht er die gewonnene Flexibilität des Unternehmens, welches eine SOA aufgebaut hat. Eine möglichst kurze „Time to market“ wird dabei immer primär angestrebt. Immer schneller muss ein Unternehmen auf sich ändernde Marktverhältnisse reagieren können. Dazu muss SOA Faktoren wie Organisation des Unternehmens, die Regeln und natürlich die Prozesse (siehe Abschnitt 2.1) mit einbeziehen. Weiter charakterisiert er SOA als ein Konzept, welches eine verteilte, heterogene Systemlandschaft mit ebenso verteilten Zuständigkeiten und Besitzverhältnissen beinhaltet. Die zentralen technischen Konzepte innerhalb von SOA sind bei ihm 1. die Services selbst, 2. eine hohe Interoperabilität der Services und 3. eine lose Kopplung der Services. Mit loser Kopplung ist hier gemeint, dass besonders die Punkte Flexibilität, Skalierbarkeit und Fehlertoleranz zu beachten sind.

Papazoglou [Pap07] definiert:

> „(SOA is) a logical way of designing a software system to provide services to either end-user applications or to other services distributed over a network, via published and discoverable interfaces.“

Hier wird also das Netzwerk betont und dass es veröffentlichte Schnittstellen gibt, die aufgefunden werden können. Dies sind typische Eigenschaften von Web-Services. Tatsächlich lässt sich Papazoglou der Web-Service-Community zuordnen. Er selbst schreibt im oben referenzierten Paper in einer Fußnote:

> „In this paper we shall use the terms Web service and service interchangeably.“

In dieser Arbeit spielen Web-Services[14] zwar eine große Rolle, stellen jedoch bei weitem nicht die einzige Möglichkeit für den Aufbau einer SOA dar.

Veröffentlichungen aus der Webservice-Community stellen bei der Definition des SOA-Begriffs oft das so genannte „SOA-Dreieck“ (oder Webservices-Dreieck, siehe Abbildung 2.3) in den Mittelpunkt. Hier existieren drei kommunizierende Parteien:

1. Ein **Service-Provider**, welcher einen Service anbieten möchte,
2. eine **Service Registry**, welche die Menge aller öffentlichen Dienste verwaltet und

[14] besonders durch WSDL beschriebene SOAP-Web-Services

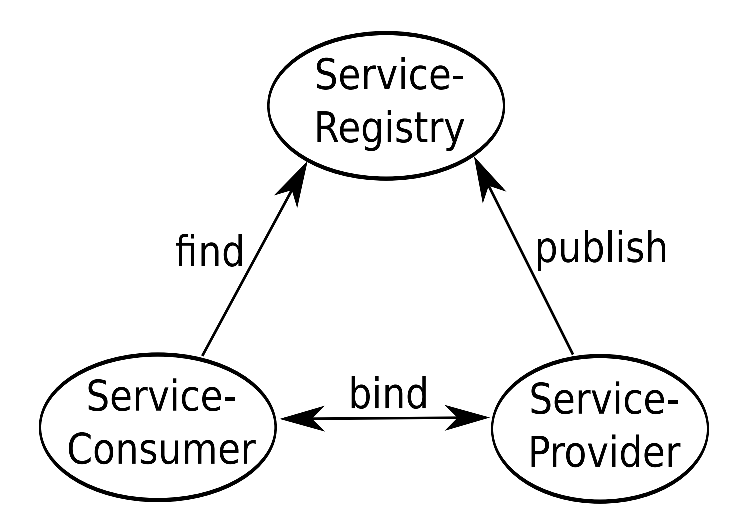

Abbildung 2.3: Das so genannte „SOA-Dreieck“

3. ein **Service-Consumer**, welcher einen Dienst aufrufen möchte.

Zwischen diesen Parteien existieren nun drei typische Nachrichtenarten:

1. **publish (veröffentlichen)**: Der Service-Provider registriert den Service in der Service-Registry
2. **find (finden)**: Der Service-Consumer sucht in der Service-Registry nach dem richtigen Dienst
3. **bind/interact (verbinden/interagieren)**: Service-Provider und Service-Consumer kommen zusammen. Der Consumer kann den Service aufrufen, d.h. er kann nun dem Provider eine Anfragenachricht schicken, welcher mit einer Antwortnachricht reagiert.

Um SOA und Webservices auseinanderzuhalten, benutzt das W3C bei der Beschreibung dieses Dreiecks den Begriff „Web Services Architecture“ [W3C02] und nicht „SOA“. Obwohl beide Begriffe nicht deckungsgleich, so ist es doch so, dass Webservices die dominierende Technologie zur Umsetzung serviceorientierter Architekturen darstellen. Dabei wird für die Service-Definition beim Service-Provider WSDL[15] eingesetzt, für das Veröffentlichen bei einer Registry UDDI[16] und für die Kommunikation zwischen Consumer und Provider SOAP. Der Registry-Teil fällt heute oft weg oder wird anders gelöst. UDDI hat sich vor allem aufgrund seiner Komplexität nicht durchsetzen können. Öffentliche UDDI-Server im Internet existierten nur für eine sehr kurze Zeit und die aktuelle Version der W3C-Spezifikation enthält nur noch den Consumer und den Provider, das Dreieck wurde also zu einer Zwei-Parteien-Kommunikation

[15]WSDL: Web Services Description Language
[16]UDDI: Universal Description, Discovery and Integration

[W3C04d]. Trotzdem sind Registrys immer noch wichtig, wenn auch oft nur innerhalb eines Produktes oder innerhalb eines Unternehmens. In Kapitel 3 wird für die untersuchten Service-Sammlungen genau dargelegt, ob und wie eine Registry jeweils umgesetzt wurde.

2.3 Application Programming Interfaces (APIs)

In Abschnitt 2.2 wurde erläutert, was Services sind und was eine SOA ist. Services treten dabei selten allein auf, sondern sind meist Teil ganzer Service-Sammlungen. Die Schnittstellenbeschreibungen dieser Service-Sammlungen heißen APIs („Application Programming Interfaces") und erreichen (gerade bei Enterprise-Systemen) oft einen erheblichen Umfang. Der API-Begriff ist dabei natürlich älter als das SOA-Thema. Eine API existiert immer dann, wenn Schnittstellenbeschreibungen vorliegen, so dass verschiedene Softwarekomponenten miteinander interagieren können. Genau in diese Richtung geht eine Definition aus einem Paper von de Souza et al.[SRC+04], bei der sie sich auf Aussagen aus einer Präsentation von des Rivieres [Riv04] beziehen:

> „An API is a well-defined interface, usually supported by the underlying programming language, that allows one software component to access programmatically another component."

De Souza et al. setzen dem die etwas formalere Definition des Carnegie-Mellon-Software-Engineering-Institute entgegen [Car03]:

> „Application Programming Interface (API) is an older technology that facilitates exchanging messages or data between two or more different software applications. API is the virtual interface between two interworking software functions such as a word processor and a spreadsheet. (...) An API is the software that is used to support system-level integration of multiple commercial-off-the-shelf (COTS) software products or newly-developed software into existing or new applications."

Hier wird betont, dass eine API die Schnittstelle zwischen zwei (oder mehr) Software-Systemen darstellt. De Souza et al. fügen letztlich noch eine eigene, pragmatische Definition hinzu:

> „(...) any well defined interface that defines the service that one component, module or application provides to other software elements."

Diese Definition kommt dem heutigen API-Begriff im SOA-Umfeld wohl am nächsten, da hier im Zentrum steht, dass ein Software-Konstrukt seine Funktionen für die Außenwelt beschreibt. Die Gesamtheit der beschriebenen Funktionen ist also das, was die Software „anbietet“. Dieser „Angebotscharakter“ ist ein grundlegendes Konzept von APIs. Clarke [Cla04] führt an, dass hierfür in der Psychologie der Begriff „Affordanz“ (engl. „affordance“) geprägt wurde. Dieser geht zurück auf James J. Gibson [Gib77], wonach Objekte eine bestimmte „Affordanz“ besitzen also einen „Angebotscharakter“. Zum Beispiel bietet ein Stuhl an, dass man sich darauf setzt und eine Türklinke, dass man sie herunter drückt, um die Tür zu öffnen. Es ist dabei wichtig, dass ein „Benutzer“ auch immer erkennt, was angeboten wird. Es ist grundsätzlich schlecht, wenn ein Angebot existiert, es jedoch nicht wahrgenommen wird. Clarke [Cla04] nennt dazu ein Beispiel aus dem GUI-Design und zwar den Start-Knopf in Microsoft Windows, dem man nicht unbedingt ansieht, dass man darauf klicken muss, um das System herunterzufahren. Ein gutes, konsistentes, möglichst einfaches und intuitives Design mit gut gewählten Begriffen und einer klaren Struktur ist also unbedingte Voraussetzung dafür, dass Benutzer das Angebot der Funktionen richtig und vollständig wahrnehmen. Dies ist ein wichtiger Grundgedanke bei einer Untersuchung von APIs, wie sie in Kapitel 3 beschrieben wird.

Wichtig ist bei APIs, dass sie Informationen verstecken. Das Prinzip der „Kapselung“ (oder auch „Black-Box-Prinzip“) ist das Mittel der Wahl, wenn große, komplexe Systeme kreiert werden sollen [Par72]. Nur durch Modularisierung und die damit zusammenhängende Einführung von Schnittstellen zwischen den Modulen können große Systeme beherrscht werden. Heutige service-orientierte Architekturen, gerade im Enterprise-Umfeld, sind dabei gute Beispiele für extrem komplexe Systeme. Aber auch schon bei der Einführung von Subroutinen (oder Funktionen, Prozeduren, ...) und später der Klassen und Objekte in der objektorientierten Programmierung war Kapselung und damit Abstraktion ein entscheidender Faktor. Die Aufteilung einer Software in bestimmte Einzelteile [17] führt auch dazu, dass eine gewisse Arbeitsteilung innerhalb einer Gruppe von Entwicklern möglich ist. Erst so können wirklich große Gruppen von Personen an einem Produkt gemeinsam arbeiten. APIs dienen demnach nicht nur als Strukturierungs- und Beschreibungsmittel der Software sondern auch als Organisationsmittel unter Softwareentwicklern [SRC+04]. Teambildung und API-Struktur beeinflussen sich gegenseitig. Die Struktur der APIs beeinflussen wie Teams gebildet werden müssen und jedes Team definiert wieder für sich APIs als Schnittstelle zu den anderen Teams. Die Gesetzmäßigkeit dieses Zusammenhangs wird auch „Conway’s law“ genannt [SRC+04, Par72].

[17] z. B. „Module“ genannt, man könnte aber auch „Teilprojekte“ oder „Unterprojekte“ sagen

Neben dem Beherrschen einer Programmiersprache[18] ist die Kenntnis und der richtige Umgang mit APIs heute die Kernkompetenz zur Entwicklung von Software schlechthin. Standardaufgaben müssen immer seltener selbst entwickelt werden, da es für sehr viele Anwendungsfälle bereits fertige Bibliotheken, Frameworks oder Services (z. B. in der Cloud) gibt. Diese Elemente werden dabei alle über ihre APIs benutzt. Ohne den Einsatz von Fremdsoftware ist ein Softwareprodukt von relevanter Größe heute nicht mehr vorstellbar. Der Ausspruch „auf den Schultern von Riesen stehen“, der Newton zugeschrieben wird, kommt hier voll zum tragen. Es wäre deutlich aufwändiger und besonders fehleranfälliger, wenn man immer wieder die gleichen Probleme aufs Neue löst. Diesen Wandel in der Softwareentwicklung kann man mit „Surfen statt Schwimmen“[19] umschreiben. Anstatt sich abzumühen und trotzdem nur langsam voran zu kommen (schwimmen), stellt man sich auf ein Surfbrett und nutzt die vorhandenen Wellen. Natürlich muss man zunächst einmal lernen wie dies geht. Das gilt sowohl für das Surfen als auch für die Einarbeitung in fremde APIs.

Es existiert mittlerweile eine wissenschaftliche Community beachtlicher Größe, die sich mit den Themenbereichen „API-Usability“[20] bzw. „API Design“[21] beschäftigen. Diese Gebiete hängen naturgemäß stark zusammen und lassen sich nicht immer scharf trennen. Burns et al. [BFHM12] führten eine Meta-Studie über zahlreiche Veröffentlichungen in diesem Gebiet durch. Sie kategorisierten diese folgendermaßen:

1. **„Design Papers“**: Veröffentlichungen über Entwurfsregeln von APIs [Dau10, ESM07, HRH08, RKS10, SK11, SC07, SM07]
2. **„Framework Papers“**: Veröffentlichungen über Methoden zur Evaluierung von API-Qualität [Cla04, BB05, FZ10, GJZ+11, O'C10, RJ08, RJ07, SB09]
3. **„Tools Papers“**: Veröffentlichungen über Tools, welche die Benutzbarkeit von APIs verbessern [DH09, ESM10, PH09, DH09, Wat09, WMJ10]
4. **„Examination Papers“**: Veröffentlichungen, die eine spezielle Kategorie von APIs untersuchen [BJX+08, HL11, MPD10, MRTS98, NM10, Rob09, SGB+08].

Die in Kapitel 3 beschriebene Evaluation von ERP-APIs würde demnach in die zuletzt genannte Kategorie passen.

---

[18] und natürlich analytisches und logisches Denken sowie die Fähigkeit zur Abstraktion etc.

[19] Zitat Prof. Dr. Bernhard Steffen, regelmäßig in persönlicher Kommunikation, z. B. in der Abschlussbesprechung der Projektgruppe PCB („Process-Cloud for Business“) am 27.11.2012

[20] Benutzerfreundlichkeit der APIs

[21] Entwurfsregeln für APIs

Eine weitere sehr allgemeine Veröffentlichung über das gesamte Feld „API-Usability“ ist der Bericht einer entsprechenden Interessensgruppe[22] bei der CHI-Konferenz[23], welche 2009 veröffentlicht wurde [DFMS09]. Der Bericht beschreibt, was unter „API-Usability“ zu verstehen ist und welche Fragestellungen in dem Gebiet aktuell untersucht werden. Die Gruppe gründete auch eine Web-Präsenz unter www.apiusability.org, auf der Links und Ressourcen zum Thema gesammelt werden.

Zum Design von APIs stellt Bloch [Blo06] eine umfangreiche Sammlung von Tipps für Software-Entwickler zusammen. Eine zentrale Forderung ist dabei, dass eine veröffentlichte API sich im Idealfall niemals ändern sollte, da es immer noch Software geben kann, die diese API benutzt und davon abhängig ist, dass sie auch genau so funktioniert, wie sie ursprünglich erdacht und spezifiziert wurde. Der Entwurf einer API muss also immer mit besonders großer Sorgfalt geschehen und es sollte immer mehrmals geprüft und überdacht werden, ob der aktuelle Stand „reif“ genug ist, veröffentlicht zu werden. Die besondere Wichtigkeit von API-Stabilität und die damit zusammenhängende Herausforderung eines gut funktionierenden Versionierungskonzeptes werden in Kapitel 3 wieder aufgegriffen, wenn für die Untersuchung von ERP-APIs genau definiert wird, welche Kriterien für diese Kategorie von APIs besonders zu beachten sind. Weitere erwähnte Punkte sind zum Beispiel, dass eine API einfach zu lesen, zu lernen und zu benutzen sein sollte und falsche Benutzung möglichst unwahrscheinlich ist, die API hinreichend mächtig ist um alle Anforderungen zu erfüllen. Außerdem sollte die API einfach zu erweitern sein und für die Zielgruppe (also die Benutzer der API) angemessen entworfen worden sein. Diese Punkte haben jeweils auch ihre Entsprechungen in Abschnitt 3.1, in dem die Kriterien für gute ERP-APIs genannt werden.

Die meisten Verfahren, um zu ermitteln, wie gut eine API entworfen worden ist, stammen aus dem Bereich der Mensch-Maschine-Interaktion (MMI)[24] [GPT12]. Die prominenteste Methodik ist hier das „Think-aloud-protocol“ [ES85], also ein schriftliches Protokoll der laut ausgesprochenen Gedanken eines API-Benutzers. Einer Menge von Probanden wird dazu die Aufgabe gestellt, eine kleine Software zu programmieren, die durch die Benutzung der zu untersuchenden API ein vorgegebenes Problem lösen soll. Durch die Protokollierung der Gedanken kann dann festgehalten werden, wo genau die „Stolpersteine“ in der API liegen. In dieser Arbeit werden APIs jedoch nicht nur aus dem Blickwinkel der Mensch-Maschine-Interaktion, sondern auch aus dem der Maschine-Maschine-Interaktion gesehen. Das liegt daran, dass eine API nicht nur

[22] mit dem Namen „API usability“
[23] International Conference on Human Factors in Computing
[24] auf englisch: „Human-Computer-Interaction“ (HCI)

dazu genutzt werden kann, um von Hand Code zu erstellen, welche die damit beschriebenen Funktionen nutzt, sondern auch um sie automatisch durch Software auswerten zu lassen um dann Code zu generieren, der wiederum die API nutzt. Der Mensch kommt also nicht mehr unmittelbar mit der API in Berührung sondern nur noch mittelbar. Eine maschinenlesbare API ist also - wie zum Beispiel im Bereich des „Semantic Web“ [BLHL+01, SHBL06] - das Ziel. Unsere Erfahrung hat gezeigt, dass sich jedoch die konkreten Technologien des Semantic Web (wie zum Beispiel RDF [W3C04c], RDF-Schema [W3C04b], OWL [W3C12] oder OWL-S [W3C04a]) im Enterprise-Umfeld nicht durchgesetzt haben. Keine ERP-API nutzte eine dieser Technologien. Es ist zu vermuten, dass dies daran liegt, dass der Mehraufwand für eine solche API-Definition ungleich höher ist als bei einer traditionellen. Dies ist auch das Hauptargument in einem Blog-Eintrag von Brian Huff [Huf09] von 2009 mit dem Titel „The Semantic Web: Impossible In Theory, Impractical in Reality“. Schon zwei Jahre vorher veröffentlichte Stephen Downes in seinem Blog den Artikel „Why the Semantic Web Will Fail“ [Dow07]. Sein Hauptargument für die fehlende Akzeptanz der Semantic-Web-Technologien und den Misserfolg des Semantic Web im Allgemeinen ist, dass das Hauptziel der Service-Anbieter, welche meist Weltkonzerne sind, Gewinnmaximierung und nicht Interoperabilität ist. Daher sei es meist gar nicht gewollt, dass sich alle Anbieter auf einen Standard einigen und hinreichend Informationen mit ihrer API veröffentlichen, so dass sich alles möglichst automatisch integrieren lässt.

Um eine API zu benutzen ist es natürlich unabdingbar, dass dem Benutzer die zu Grunde liegenden Konzepte bekannt sind, dass er also weiß, wie die einzelnen Funktionen zusammenhängen und was man überhaupt damit machen kann. Ko und Riche [KR11] schrieben über die Wichtigkeit des konzeptuellen Wissens für das Verständnis und die Benutzung einer API. Sie fordern, dass eine API-Dokumentation nicht nur Beschreibungen aller Klassen, Methoden und Parameter sondern auch eine allgemeinere Einführung mit fundamentalen Grundlagen enthalten soll. Für die APIs, die in Kapitel 3 untersucht werden, würde dies bedeuten, dass grundlegende kaufmännische Kenntnisse sowie Wissen über das jeweilige ERP-System und dessen Struktur in der API-Dokumentation thematisch behandelt werden. In einem sehr eingeschränkten Maße könnte dies natürlich in manchen Punkten helfen, der Aufwand würde jedoch schnell in keinem vernünftigen Verhältnis zum Nutzen stehen. Man sollte nicht versuchen, ganze ERP-Handbücher oder sogar komplette Studieninhalte in einer API-Dokumentation unterzubringen.

Oft ist die Benutzung einer öffentlichen API nicht der einzige Weg, um an ein System von außen heranzukommen. Neben dem „offiziellen“ Weg gibt es fast immer auch eine Art „Hintertür“. Bei ERP-APIs wäre dies zum

Beispiel der direkte Zugriff auf die hinter dem ERP-System liegenden relationalen Datenbanken über SQL. Dies hat jedoch deutliche Nachteile gegenüber einer API. Es ist hier grundsätzlich eine extrem genaue Kenntnis der Datenstruktur erforderlich und sämtliche Abhängigkeiten müssen beachtet werden, um nicht einen inkonsistenten Datenbestand zu erzeugen. Ebenso werden hier keine Geschäftsregeln automatisch beachtet (wie es bei der API-Benutzung der Fall wäre), so dass man nie sicher ist, ob man im Sinne des Unternehmens und dessen Regeln handelt. Das wohl schwerwiegendste Problem ist jedoch, dass sich das interne Datenschema auch einfach irgendwann ändern kann und nicht wie bei der API (im Idealfall) stabil bleibt. Es kann also zu jeder Zeit sein, dass so eine Lösung nicht mehr funktioniert. Businge et al. [BSB12] nennen solche Hintertüren „bad non-APIs“ im Unterschied zu den offiziellen „good APIs“. Sie beschäftigten sich mit der Benutzung der Eclipse-API in Eclipse-Plugins, und zwar untersuchten sie, inwieweit Entwickler die veröffentlichte API oder stattdessen eine „bad non-API“ (also in diesem Fall die nicht veröffentlichten, internen Klassen und Methoden) nutzten. Sie fanden heraus, dass 44% der analysierten Plugins non-APIs benutzten. Dies ist in dem Bereich schon eine erschreckende Zahl, wäre jedoch im ERP-API-Bereich noch weit schlimmer.

Bei der Vielzahl von Veröffentlichungen zum Thema „API-Usability“ auf der einen Seite und der Wichtigkeit von APIs im SOA-Umfeld ist es umso erstaunlicher, dass bis jetzt recht wenige Arbeiten gibt, die beide Aspekte im Zusammenhang untersuchen. Maleshkova et al. [MPD10] untersuchten 2010 „Remote-APIs“, also APIs von entfernten Systemen, konzentrierten sich dabei jedoch auf die APIs von Web-Applikationen wie zum Beispiel Flickr, del.icio.us oder Doodle. Insgesamt wurden mehr als 200 Web-Applikationen analysiert, die alle im Verzeichnis programmableweb.com gelistet sind, Statistiken angefertigt und folgende Aspekte untersucht:

- die Größe der APIs (Anzahl von Operationen)
- die technische Basis (REST[25] oder RPC[26])
- die Benutzung in Mash-ups
- ob Default-Werte für Parameter vergeben sind
- ob optionale Parameter existieren (die auch so gekennzeichnet sind)
- ob es „coded parameters“ gibt, also Parameter, deren Wert von einer Berechnung abhängt
- ob alternative Werte unterstützt werden (Auswahllisten)

[25]REST: Restfull State Transfer
[26]RPC: Remote Procedure Call

- welches Format unterstützt wird (XML[27], JSON[28],...)
- wie genau die Aufrufdetails aussehen (z. B. zur Authentifizierung)
- ob eine gute Dokumentation mit Beispielen und Listen von Fehlernachrichten und -codes existiert

Insgesamt kommen sie zu der Erkenntnis, dass bei REST-APIs oft das Problem der „Unterspezifizierung“ vorliegt, dass also zu wenige Informationen über den Service nach außen kommuniziert werden. So wird oft nicht einmal genau ersichtlich, was genau an Eingabedaten verlangt wird oder was als Ausgabedaten produziert wird. So extrem kann dieses Problem bei SOAP-basierten APIs nicht auftreten, da in einer WSDL die Schnittstelle immer mit der Hilfe von XSD genau definiert werden muss. Weiterhin kritisieren sie APIs, die zwar direkt HTTP benutzen, sich jedoch nicht an das REST-Paradigma halten. Zum Beispiel ändert die GET-Methode in manchen Fällen Daten oder die POST-Methode wird benutzt um Daten zu holen. In dieser Studie wurden viele APIs untersucht (ohne sich auf ein bestimmtes Gebiet zu fokussieren), der Großteil der APIs sind jedoch kleine APIs mit sehr wenigen Operationen. Dies ist ein wesentlicher Unterschied zu den ERP-APIs, die in dieser Arbeit im Mittelpunkt stehen.

Es existieren noch drei recht ähnliche (etwas ältere) Studien über Web-APIs, die 2005 von Fan und Kambhampati [FK05], 2007 von Li et al. [LLZ+07] und 2008 von Al-Masri und Mahmoud [AMM08] veröffentlicht wurden. Die erste betont hauptsächlich die geringe Größe der APIs („more than 77% of the services have less than 5 operations“), die fehlende Dokumentation, besonders in den WSDL-Dokumenten und dass die meisten Services lediglich zum Auslesen von Daten benutzt werden. Die zweite Studie, die zu sehr ähnlichen Ergebnissen kommt, erwähnt ebenfalls die geringe durchschnittliche API-Größe. Die einzige komplexe API ist jene der bekannten Verkaufs- und Handelsplattform eBay. Dies zeigt den typischen Charakter von Business-APIs: Die Anzahl der Operationen ist relativ hoch und die Datenstrukturen weisen eine komplexe Struktur auf. Die dritte Studie legt seinen Fokus auf die Qualität der APIs, jedoch hauptsächlich auf einfache statistische Werte wie die folgenden:

- Ist der Service wirklich erreichbar?
- Existiert ein valides WSDL-Dokument?
- Wie groß ist das WSDL-Dokument?
- Welche Programmiersprache wurde für die Implementierung der Services genutzt?

[27]XML: Extensible Markup Language
[28]JSON: JavaScript Object Notation

Wenn man das Gebiet der Remote-API-Untersuchungen noch auf den Business-Bereich (also im Wesentlichen auf ERP-APIs) einschränkt, so wird die Zahl der Veröffentlichungen noch dramatisch kleiner. Borovskiy et al. [BZKP09] kritisieren das schlechte Design der APIs heutiger ERP-Systeme stark. Als Alternative schlagen Sie die Benutzung einer Abfragesprache „Business Object Query Language“) vor, mit denen auf die Business-Objekte des ERP-Systems zugegriffen werden soll. Die Kritik an mangelnder Qualität ist besonders daher sehr bemerkenswert, da einer der Co-Autoren dieser Veröffentlichung Hasso Plattner ist, Mitbegründer und Mitglied des Aufsichtsrates von SAP, dem weltweit führenden Hersteller von ERP-Systemen.

Beaton et al. [BJX+08, BMS+08] beschäftigen sich mit SAPs eSOA-Technologie und deren Analyse mittels Verfahren aus dem Bereich der Mensch-Maschine-Interaktion. Sie präsentieren auch einen entsprechenden Ansatz zur API-Evaluierung vor. Sie kritisieren die sehr komplexen Datenstrukturen, die vage Dokumentation, die langen Service-Namen und die zahlreichen Inkonsistenzen in der API. In einer Fallstudie mit einer Gruppe von sechs Studenten hat es kein einziger Teilnehmer geschafft, den Client-Code für einen recht einfachen Service zu implementieren. In anderen Publikationen konzentrieren Beaton et al. sich komplett auf die eSOA-Dokumentation [JXB+09, MJX+10].

2.4 Cloud-Computing

These T2 (siehe Abschnitt 1.2) besagt, dass durch Cloud-Computing insgesamt die Qualität der APIs besser geworden ist. Daher wird in diesem Abschnitt zunächst einmal eingeführt, was genau unter dem Begriff „Cloud-Computing“ zu verstehen ist.

Die wohl am häufigsten zitierte Definition des Begriffs „Cloud-Computing“ stammt vom amerikanischen NIST[29]. Dieses Institut definiert Cloud-Computing folgendermaßen [MG11]:

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“

Es geht also um Ressourcen, die im Netzwerk (also normalerweise im Internet) bereitgestellt werden und die jeder ohne viel Eigenaufwand

[29]NIST steht für „National Institute of Standards and Technology“ und ist Teil des Handelsministeriums der USA

einfach nutzen kann oder die Nutzung auch wieder beenden kann. Da die Software ins Netz ausgelagert wird und somit ein „Outsourcing“ des IT-Betriebs vorgenommen wird, minimiert man den eigenen Aufwand und die damit zusammenhängenden Kosten, die bei einem eigenen Rechenzentrum oft wesentlich höher ausfallen können. Ein Unternehmen, welches einen Dienst in die Cloud auslagert, muss dafür keinen Administrator mehr beschäftigen, keine Hardware einkaufen und betreiben (die auch noch gekühlt werden müsste) und sich allgemein keine Gedanken mehr über technische Details wie Ausfallsicherheit, Backups oder Zugriffssicherheit machen. Die wichtigen Stichwörter heißen hier „Virtualisierung“ und „Abstraktion“.

Um genauer zu spezifizieren, was mit „Cloud-Computing“ gemeint ist, führt das NIST fünf zentrale Eigenschaften von Cloud-Computing auf:

1. **On-demand self-service** (Selbstbedienung nach Bedarf): Ein Benutzer kann jederzeit selbst bestimmen, welche Ressourcen genutzt oder nicht genutzt werden. Dazu ist keine menschliche Interaktion mit dem Service-Provider nötig.
2. **Broad network access** (Breitbandiger Netzwerkzugang): Der Dienst wird im Netz (typischerweise das Internet) angeboten und zwar über Standard-Technologien.
3. **Resource pooling** (Zusammenlegung der Ressourcen in Pools): Die physikalischen Ressourcen (Rechner, Speicherplatz, etc.) werden nicht direkt einem Benutzer zugewiesen, sondern in einem Pool verwaltet, der von allen Benutzern gemeinsam genutzt wird. Von der realen Hardware wird abstrahiert. Ein Benutzer weiß nicht (und muss auch nicht wissen), auf welcher Hardware sein Anwendungsfall im Moment ausgeführt wird.
4. **Rapid elasticity** (Sofortige Elastizität): Alle Cloud-Angebote können sofort (d. h. ohne Verzögerung) bereitgestellt und auch wieder heruntergefahren werden. Für den Benutzer sollte das Angebot somit unbegrenzt erscheinen.
5. **Measured service** (Gemessene Dienstnutzung): Es wird immer genau gemessen, wie viel jeder Dienst von welchem Benutzer eingefordert wurde. Dadurch ist es möglich, dass ein Kunde immer nur genau das zahlt, was er auch „verbraucht“ hat.

Man kann Cloud-Computing also als Rechen- und Speicherleistung „aus der Steckdose“ bezeichnen, da es vergleichbar ist mit der heutigen häuslichen Versorgung mit Strom, Wasser und Gas. Alles ist prinzipiell unbeschränkt verfügbar, man bezahlt genau das, was man verbraucht hat und es funktioniert einfach ohne dass man sich selbst um die dahinter liegende Technologie kümmert (in dem Beispiel also z. B. das Reinigen

des Wassers in der Kläranlage, die Gewinnung von elektrischem Strom durch Kohle, Gas, Wind, Sonne, etc. oder die Förderung von Erdgas). Diese Idee ist bei weitem nicht neu. Schon 1961 sagte John McCarthy[30] bei der Einhunderjahrfeier des MIT[31] [Abe99]:

> „If computers of the kind I have advocated become the computers of the future, then **computing may someday be organized as a public utility** just as the telephone system is a public utility. ... The computer utility could become the basis of a new and important industry."

Zu der Zeit war es üblich, dass man große Mainframes einsetzte, auf die man mit schlanken Clients (meist so genannte „Terminals") zugriff. Auch hier teilten sich viele Nutzer eine gemeinsame Ressource im Netzwerk. Als die Rechner immer kleiner und leistungsstärker wurden, kam es dann irgendwann zur „PC-Ära", in der es für jeden möglich war, seine eigenen Computer zu benutzen. Somit wurde die Vision von John McCarthy zunächst einmal immer unwahrscheinlicher. Auch in den Anfangsjahren des Internet in den 90er-Jahren war aufgrund geringer Datenübertragungsraten noch nicht an Cloud-Computing zu denken. In den 2000ern kam dann zuerst das Prinzip der „Application Service Provider" (ASP) auf. Dabei wurden Applikationen nicht mehr lokal auf einem Desktop-Rechner installiert und ausgeführt, sondern liefen als Web-Applikation und somit in jedem Browser. Man musste die Software also nicht mehr selbst installieren und sich nicht um Updates kümmern. Natürlich kamen die GUIs anfangs bei weitem nicht an die der Desktop-Applikationen heran. Dies wurde jedoch mit schnelleren Internetanschlüssen und besonders mit dem Aufkommen der AJAX-Technologie[32] immer besser. Der ASP-Ansatz zeigte schon einen wichtigen Teil von dem, was heute „Cloud-Computing" genannt wird, dieser Begriff wurde zu dem Zeitpunkt jedoch noch nicht genutzt.

Im Jahr 2006 kam mit Amazon der große Durchbruch des „Cloud"-Begriffs. Der Online-Handel („E-Commerce") erlangte im Vergleich zum klassischen Einzelhandel eine immer größere Bedeutung [Sol05] und Amazon selbst hatte sich von einem Online-Buch-Händler zu einem der größten Player in diesem Bereich entwickelt [fab11]. Um so einen Dienst zu betreiben, hatte das Unternehmen ein beachtliches Rechenzentrum aufgebaut, welches jedoch nicht immer gleichmäßig ausgelastet war. Zum Beispiel wurde zu Weihnachten besonders viel Rechenleistung benötigt,

[30] Turing-Award-Gewinner und Erfinder von LISP
[31] Massachusetts Institute of Technology in Cambridge, Massachusetts, USA
[32] „Asynchronous JavaScript and XML": eine Kombination der Technologien JavaScript und XML (oft auch JSON, also JavaScript Object Notation, statt XML) um einer Webseite asynchron mit dem Webserver zu kommunizieren, und dann per DOM-Manipulation (Document Object Model) nur Teile der Seite zu ändern anstatt eine ganze Seite neu zu laden.

um mit den Kundenanstürmen zurechtzukommen und zu manch anderen Zeiten war es deutlich ruhiger. Da die brach liegenden Ressourcen genutzt werden sollten, entschied sich Amazon dazu, diese an die Allgemeinheit zu vermieten. Nun konnte jeder die Rechen- und Speicherleistung von Amazon nutzen. Das Angebot existiert unter der Bezeichnung „Amazon Web Services“[33] [Ama13] und umfasst mittlerweile eine Vielzahl an verschiedenen Angeboten.

Bis hierhin wurde sehr deutlich, dass extrem unterschiedliche Ansätze als „Cloud-Computing“ bezeichnet werden. Auf der einen Seite gibt es die ASP-Angebote, auf der anderen Seite die Rechen- und Speicherdienste von z. B. Amazon. Und später kamen noch viele weitere, wieder andere Dienste hinzu, die ebenfalls unter dem Stichwort „Cloud“ geführt werden. Es muss also eine Kategorisierung der unterschiedlichen Dienste erfolgen. Dies ist auch geschehen. Im oben erwähnten NIST-Dokument [MG11] wird dargelegt, dass sich im Wesentlichen die Kategorien SaaS („Software as a Service“), PaaS („Platform as a Service“) und IaaS („Infrastructure as a Service“) ausmachen lassen.

- **SaaS**: In die Kategorie „SaaS“ fallen alle Dienste, die vor der Cloud-Ära als „ASP“ bezeichnet wurden, also fertige Web-Applikationen, die unter anderem dank AJAX sehr nah an Bedienbarkeit und das Feature-Reichtum von Desktop-Applikationen heran reichen. Ein prominentes Beispiel für eine erfolgreiche SaaS-Lösung ist Salesforce, eine Cloud-CRM-Lösung[34]. Amazon und Salesforce sind zusammen die Vorreiter im Bereich Cloud-Computing, die diesen Begriff wesentlich geprägt haben.
- **IaaS**: In diese Kategorie gehören z. B. die Dienste von Amazon. Hier werden Ressourcen wie Rechenleistung oder Speicherplatz „in der Cloud“ angeboten. Dies sieht zum Beispiel so aus, dass man beliebige virtuelle Maschinen deployen kann, auf denen man dann selbst wieder beliebige Software installieren kann.
- **PaaS**: Zwischen den beiden schon genannten Kategorien entwickelte sich etwas später mit „PaaS“ eine weitere Kategorie, bei der nicht die nackte Infrastruktur angeboten wurde, aber auch keine fertige Anwendung, sondern eine bestimmte Platform, auf deren Basis eigene Applikationen entwickelt und deployt werden können. Ein bekanntes Beispiel hierfür ist die „Google App Engine“, auf der man eigene Software[35] deployen kann ohne sich Gedanken über ein Betriebssystem oder andere technische Hintergründe zu machen. Auch Salesforce hat mit Force.com ein PaaS-Angebot.

[33] oder kurz: „Amazon WS“
[34] CRM: Customer Relationship Management (Verwaltung von Kundenbeziehungen)
[35] in Python, Java oder Go implementiert

Es existiert noch eine weitere Dimension, wie man Cloud-Dienste kategorisieren kann und zwar das „Deployment-Modell“. Auch dies wird vom NIST aufgegriffen [MG11]:

- **Private cloud**: Die Cloud-Anwendung steht exklusiv einem Unternehmen zur Verfügung.
- **Community cloud**: Die Cloud-Anwendung steht einem Zusammenschluss verschiedener Unternehmen mit einem gemeinsamen Interesse zur Verfügung.
- **Public cloud**: Die Cloud-Anwendung steht der gesamten Öffentlichkeit zur Verfügung. Dies ist wohl die bekannteste und am weitesten verbreitete Variante.
- **Hybrid cloud**: Alle möglichen Kombinationen aus den drei obigen Deployment-Modellen fallen unter die Kategorie „Hybrid cloud“.

Mit dem Aufkommen des Cloud-Computings wächst sowohl insgesamt die Anzahl der existierenden APIs (siehe Abschnitt 2.3) gewaltig als auch deren Bedeutung. In einer Einladung zu einem Web-basierten Seminar des ESB-Herstellers „MuleSoft“ [Mul13a] mit dem Titel „Hybrid Cloud Architecture is Coming: Are You Ready?“ wird hierfür der treffende Begriff „API-Explosion“ gebraucht. Eine Software, die in der Cloud (z. B. als SaaS) angeboten wird, ist quasi verpflichtet, neben der Web-GUI auch eine API anzubieten, und diese sollte möglichst von hoher Qualität sein. Nasser [Nas] betont die zentrale Rolle von APIs im Bereich des Cloud-Computing und fordert eine agile Herangehensweise bei deren Entwicklung, wobei also nicht die API von vornherein geplant sein sollte, sondern sich iterativ aus Benutzerbedürfnissen heraus entwickeln sollte. Dies steht natürlich teilweise im Widerspruch zu der in Abschnitt 2.3 geforderten Eigenschaft, dass sich APIs möglichst nicht verändern sollten. Durch ein gutes Versionierungskonzept ist es jedoch möglich, diese scheinbaren Gegensätze (jedenfalls zu einem gewissen Grad) zu vereinbaren. Dies wird später in Abschnitt 3.1 beschrieben.

Da die Software-Integration durch eine API für ein SaaS-System von noch größerer Bedeutung ist als für traditionelle[36] Systeme, existieren mehrere Veröffentlichungen, die sich mit der Integration im Bereich SaaS befassen. Hai und Sakoda [HS09] schreiben über „Best Practises“ in diesem Bereich, Liu et al. [LGZC10] haben dabei eine eher technische Sicht auf das Thema und schlagen eine konkrete Architektur für SaaS-Integration vor („SaaS Adapter Framework“). In [LLC09] konzentrieren sie sich auf die Frage, wie eine Firewall in SaaS-Integration-Szenarien überwunden werden kann. Shi [Shi12] schrieb seine Master-Arbeit über

[36] auch „on-premise“, also in etwa „auf dem Firmengelände“

die Integration traditioneller ERP-Systeme mit Cloud-Diensten und führte dafür einige Interviews mit Personen aus der Industrie durch.

Leider kann die Existenz von APIs für Cloud-Anwendungen jedoch auch Probleme bereiten. Dies ist zum Beispiel dann der Fall, wenn das Angebot in Web prinzipiell kostenlos ist und sich das betreibende Unternehmen irgendwann überlegt, wie es mit dem Dienst Geld verdienen soll. Die naheliegendste Lösung ist hier fast immer das Schalten von Werbung in der Web-Oberfläche. Diese Werbung können dann natürlich nur die GUI-Nutzer sehen. Wenn jemand über Fremdsoftware (z. B. eine App für ein mobiles Endgerät), das den Dienst über dessen API nutzt, auf das Angebot zugreift, fällt die Werbung natürlich weg. Das wiederum kann dazu führen, dass APIs extrem eingeschränkt werden. Das prominenteste Beispiel hierfür ist wohl die Twitter-API [Twi13], die es Fremdsoftware immer schwieriger macht, auf deren Dienst zuzugreifen [Luc12].

Insgesamt gibt es beim Cloud-Computing noch weitere Herausforderungen, die gemeistert werden müssen [DWC10]. Die schwerwiegendste ist wohl das Problem des oft fehlenden Vertrauens gegenüber einem Cloud-Anbieter. Ein Kunde kann nie ganz sicher sein, was ein Cloud-Anbieter mit den Daten macht. Wichtige und vertrauliche Daten möchten Unternehmen oft nur ungern in der Cloud platzieren. Besonders wenn Daten in den USA gespeichert werden, gibt es aufgrund des „Patriot Act“ das Problem, dass die US-Regierung im Sinne der Terrorismusbekämpfung das Recht hat, auf die Daten zuzugreifen.

Ein weiteres Problem ist die Ausfallsicherheit. Es ist schon häufiger vorgekommen, dass technische Probleme dazu geführt haben, dass ein Cloud-Dienst zeitweise nicht zugreifbar war. In dem Fall hat man als Kunde keine andere Wahl als zu warten, bis der Anbieter das Problem gelöst hat. Dies kann in unternehmenskritischen Situationen schnell den Verlust von viel Geld oder sogar Kundenverlust bedeuten. Daher ist die Vereinbarung von Service-Level-Agreements (SLA) unabdingbar. Darin muss genau geregelt werden, was der Cloud-Anbieter dem Kunden zusichert, so zum Beispiel eine maximale Ausfallrate. Bei einem Verstoß gegen ein SLA muss der Anbieter dann zum Beispiel für die Umsatzeinbußen des Kunden aufkommen. Da es sehr unterschiedliche Cloud-Modelle gibt, muss jeweils einzeln genau überlegt werden, wie ein entsprechendes SLA aussieht.

Wenn die Ausfallsicherheit ein so wichtiges Thema ist, so stellt sich natürlich auch die Frage, was ein Kunde macht, wenn er nicht einmal Zugang zum Netz hat, z. B. bei einem Kundenbesuch in einem Gebiet ohne mobilen Internetempfang. Die Offline-Benutzung ist bei den meisten Cloud-Angeboten nicht vorgesehen. Dies ist nach wie vor ein Vorteil der Desktop-Applikationen.

Die Vielzahl verschiedener Cloud-Angebote bedeutet heute leider auch eine Vielzahl verschiedenster Konzepte und Technologien. Es gibt kaum Standards, weder im Bereich SaaS noch für die Plattformen bei PaaS, noch für die Bereitstellung von Rechenleistung oder Speicherplatz bei IaaS. Dies macht eine Interoperabilität zwischen Clouds schwierig bis unmöglich. Dies bedeutet im Einzelnen, dass die Migration von Software in die Cloud oder auch die Migration zwischen Clouds sehr aufwändig sein kann. Auch die Kommunikation zwischen verschiedenen Cloud-Systemen ist somit nicht geregelt [PC09].

In dieser Arbeit werden die Möglichkeiten des Cloud-Computings aus verschiedenen Blickwinkeln betrachtet. Besonders interessant sind hier die Nutzung der Cloud als Basis für den Betrieb von ERP-Systemen (siehe Abschnitt 3.3) sowie den Betrieb der BPM-Software (Modellierungstool, Engine, Aufgabenverwaltung, etc.) in der Cloud (siehe Abschnitt 8.3).

2.5 Enterprise-Resource-Planning (ERP)

Diese Arbeit behandelt das Thema der Service-Integration hauptsächlich mit der Integration von ERP-Systemen. ERP steht dabei für „Enterprise Resource Planning“ also wörtlich übersetzt „Unternehmensressourcenplanung“. Ressourcen sind dabei Kapital, Betriebsmittel und Personal.

Jacobs und Weston beziehen sich auf die Definition aus dem APICS Dictionary[37]:

> „Framework for organizing, defining, and standardizing the business processes necessary to effectively plan and control an organization so the organization can use its internal knowledge to seek external advantage.“

Diese Definition zeigt, dass der Begriff sehr allgemein gehalten wird. Jede betriebliche Software eines gewissen Umfangs und mit einer gewissen Vielseitigkeit wird somit heute als ERP-System bezeichnet. An diese Konvention hält sich auch diese Arbeit.

Ein ERP-System ist ein zentrales betriebliches Informationssystem, meist mit einer relationalen Datenbank als Backend. Vor der Einführung solcher Lösungen war es üblich, dass in einem Unternehmen für verschiedene Zwecke jeweils eigene Softwareprodukte eingesetzt wurden. So existierte zum Beispiel eine kaufmännische Softwarelösung für die Buchführung, eine weitere für die Lagerverwaltung, noch eine weitere für den Einkauf usw. Diese Lösungen waren meist nicht miteinander kompatibel

[37] http://www.apics.org/dictionary/dictionary-information?ID=1294

und tauschten direkt keine Daten miteinander aus. Das bedeutete, dass bestimmte Daten redundant in mehreren Systemen vorlagen, was schnell zu Dateninkonsistenzen führte und einen hohen manuellen Aufwand für die Pflege der einzelnen Daten zur Folge hatte. Mit ERP-Systemen existiert eine gemeinsame Datenbank für das ganze Unternehmen, so dass zum Beispiel nicht nur die Lagerverwaltung genau weiß, welche Güter vorrätig sind, sondern auch der Vertrieb direkt Aussagen darüber treffen kann, was lieferbar ist und bei Bedarf einen Produktionsauftrag anstoßen kann. Dabei können alle diese Schritte in einem System durchgeführt werden.

ERP-Systeme bestehen aus vielen verschiedenen Bestandteilen, oft „Module“ genannt. Den Kern bildet meist ein PPS-System („Produktionsplanung und -steuerung“[38]), also ein System zur Planung von Terminen, Durchlaufzeiten, Beständen und Nutzung von Betriebsmitteln in der Produktion. In diesen Bereich fallen auch die Begriffe MRP („Material Requirements Planning“) bzw. dessen erweiterter Nachfolger MRP II („Manufacturing Resources Planning“), wo es um die Planung der für die Produktion notwendigen Materialien geht. Weitere Module befassen sich zum Beispiel mit der Finanzbuchhaltung, den Kundenbeziehungen (CRM - „Customer Relationship Management“), der Verwaltung der Mitarbeiter (HCM - „Human Capital Management“ oder HR - „Human Resources“) oder der Wertschöpfungskette von Lieferanten bis zum Kunden (SCM - „Supply Chain Management“). Oft gibt es auch separate Systeme für genau einen dieser Aspekte. Diese sind dann wiederum oft über Prozesstechnologie service-orientiert zu integrieren.

ERP-Systeme bilden heute den Kern einer jeden Unternehmenslandschaft und enthalten demnach einen Großteil aller technischen Funktionen, die in den Geschäftsprozessen des Unternehmens eine Rolle spielen. Um also Geschäftsprozesse zu automatisieren, müssen vor allem die Services der ERP-Systeme orchestriert werden. Oft bieten ERP-Systeme heute selbst eine Lösung zur Prozessautomatisierung[39]. Entsprechend des XMDD-Ansatzes (siehe dazu Abschnitt 2.6) wird jedoch angestrebt, den Prozess nicht in der ERP-Lösung zu platzieren, sondern als zentrales Steuerungselement des gesamten Unternehmens über allen Systemen zu positionieren. Es sollte also eine dünne Orchestrierungsschicht außerhalb der angesteuerten Systeme existieren, so dass zum Beispiel bei einem Wechsel des ERP-Systems die Geschäftsprozessautomatisierung nicht komplett neu hergestellt werden muss. Gerade in der heutigen Zeit, in der Unternehmenszusammenschlüsse und -übernahmen[40] häufige Ereignisse darstellen, sollte man mögliche Systemmigrationen im Auge

[38] oder englisch: MPC „Manufacturing Planning and Control

[39] z. B. SAP Business Workflow

[40] oft auch unter dem englischen Schlagwort „Mergers and Acquisitions“ geführt

behalten.

ERP-Systeme bieten meist umfangreiche APIs an (siehe dazu auch Abschnitt 2.3). Diese APIs haben folgende typische Eigenschaften:

- Größe: Die Anzahl der Services ist meist sehr groß.
- Komplexität: Die Struktur der Services und verwendeten Datenstrukturen ist meist äußerst komplex.
- Einheitliches Konzept: Die ganze API folgt einem möglichst einheitlichen Konzept, wie die Services aufgebaut sind und organisiert werden.

Aufgrund dieser Eigenschaften, die ERP-APIs von anderen APIs (wie z. B. den meisten Web-APIs, siehe 2.3) abheben und der hohen Bedeutung von ERP-Systemen, ist es sinnvoll, sich konkret mit der Integration von ERP-Systemen zu befassen und hier spezielle, möglichst gut passende Wege zu gehen. Diese Wege werden in Kapitel 5 behandelt.

Sehr typisch für ERP-Systeme ist es auch, dass sich hier ein eigenes technologisches Ökosystem mit eigener Programmiersprache, eigener Entwicklungsumgebung[41] usw. entwickelt hat. Bei SAP existiert zum Beispiel ABAP[42], bei Microsoft Dynamics NAV C/SIDE und bei Salesforce Apex. Es ist also jeweils sehr spezifisches technische Know-How für das Customizing dieser Produkte notwendig, also für die spezifische Anpassung an die Kundenbedürfnisse. Dies ist meist auch genau so beabsichtigt. So können die ERP-Hersteller ihre Berater damit beschäftigen, dieses Customizing für Kunden durchzuführen, was - gerade bei häufig wechselnden Kundenanforderungen - ein sehr einträgliches Geschäft sein kann. Innerhalb einer serviceorientierten Architektur ist es tendenziell denkbar, das Customizing nicht im ERP-System durchzuführen, sondern das ERP-System wie es ist als Backend zu benutzen und die Anpassungen in der Orchestrierungsschicht darüber zu positionieren. So wird auch der so genannte „Vendor-Lockin“ vermieden, also die Entstehung von Abhängigkeiten von einem bestimmten Software-Hersteller.

2.6 XMDD und jABC

Wie in These T4 formuliert, können der Ansatz des Extreme Model Driven Design (XMDD) und dessen Umsetzung jABC die Art und Weise der Service-Integration deutlich vereinfachen. Im Folgenden werden zunächst der allgemeine Ansatz, dann das grundlegende Prozessmodell,

[41]IDE: „Integrated Development Environment“

[42]In letzter Zeit immer mehr durch Java ergänzt, aber (noch) nicht ersetzt.

anschließend das Prozesskomponentenmodell (SIB) und schließlich das Tool jABC mit seinen Plugins beschrieben.

2.6.1 Extreme Model Driven Design (XMDD)

XMDD [MS04, MS06, MS08, MS09] steht für Extreme Model Driven Design und ist ein Ansatz, der das Ziel hat, es dem Business-Experten[43] zu ermöglichen, selbst direkt ausführbare Prozesse zu modellieren. Zum einen ist es also wichtig, dass Klarheit darüber herrscht, dass es mit dem Business-Experten und dem IT-Experten verschiedene Rollen gibt, die jeweils sehr unterschiedliche Kenntnisse und Fähigkeiten besitzen, zum anderen muss dafür gesorgt werden, dass diese Rollen permanent während des ganzen Entwicklungsprozesses miteinander kommunizieren. Es ist gerade nicht gewollt, dass zuerst die Management-Abteilung ein fachliches Modell entwirft, welches lediglich eine lose Spezifikation bestehend aus einer Sammlung von Abbildungen darstellt und anschließend die IT-Abteilung diese Spezifikation für die Grundlage manueller Implementierungen heranzieht. Stattdessen arbeiten die Rollen permanent zusammen und werden durch die Elemente von XMDD in ihrer Kommunikation besonders Unterstützt. XMDD beinhaltet dabei ein konkretes Meta-Modell für die Prozesse und ein Komponentenmodell für die Prozessbausteine. Bei XMDD handelt es sich also gleichzeitig um ein Vorgehensmodell und um ein konkretes technisches Modell zur Entwicklung prozessgesteuerter Applikationen.

XMDD ist bezüglich der Anwendungsgebiete ein sehr allgemeiner Ansatz und beschränkt sich nicht auf das Geschäftsprozessmanagement im engeren Sinne. Vielmehr können damit beliebige Applikationen entwickelt werden, bei deren Umsetzung der zu Grunde liegende Prozess im Mittelpunkt stehen soll. Neben Geschäftsprozessen wurden in der Vergangenheit zum Beispiel auch schon Prozesse für die Steuerung eingebetteter Systeme (z. B. Roboter [JKPM07] oder Mobiltelefone [Spi09]), für Strategien von Computerspielen (z. B. das Spiel „Vier Gewinnt“ [BJM09] oder für „Robocode“ [Sto10]), für Algorithmen im Bereich des Automatenlernens [RSB05, MSHM11] oder für die Definition von Code-Generatoren [JMS08, Jör13] umgesetzt.

Damit Business-Experte und IT-Experte reibungslos zusammenarbeiten können, ist es nach XMDD wichtig, dass sie dasselbe Modell benutzen und zwar genau eines. Anders als zum Beispiel bei der UML (Unified Modeling Language) gibt es bei XMDD nur genau ein Modell. Dieser Ansatz wird daher auch „One-Thing-Approach“ genannt. Wo es bei der UML für jeden Aspekt eine andere Modellart gibt, existiert hier nur

[43] im Kontext von XMDD auch oft „Anwendungsexperte“ genannt

der zentrale Prozess, der alles steuert. Dieses Prozessmodell bleibt der Mittelpunkt der Entwicklung von der ersten Idee bis zur finalen Umsetzung. Da es sich um ein Prozessmodell handelt, handelt es sich bei dem Modell um einen gerichteten Graphen, welcher den Kontrollfluss symbolisiert. Alle anderen Daten werden an den Prozessgraphen annotiert. Zur Beherrschung von Komplexität und zur Erzeugung der jeweils passenden Sichten auf das Modell ist der Prozess hierarchisch gegliedert. Das bedeutet konkret, dass ein Knoten des Graphen wieder eine Referenz auf ein Untermodell sein kann. Die Hierarchien können dabei beliebig tief sein.

Das Modell ist bei XMDD stets direkt ausführbar. Das kann durch einen in das Modellierungstool integrierten Interpreter geschehen. Ein anderer Weg der Ausführung ist die Code-Generierung. Hier wird aus dem Modell zunächst ausführbarer Programmcode generiert, welcher dann kompiliert und schließlich ausgeführt werden kann. Wichtig ist dabei, dass bewusst darauf verzichtet wird, den generierten Code manuell zu bearbeiten. Alle Änderungen werden auf Modellebene oder in den orchestrierten Services durchgeführt, nie im generierten Code. So entfällt das grundsätzlich schwierige Roundtrip-Problem.

XMDD kombiniert die Ideen verschiedener Bereiche:

- Service-Orientierung (SOA)
- Extreme Programming (XP)
- Modell-getriebene Entwicklung (MDD, model driven development), auch MDSD: Model driven software development
- Aspektorientierung (AOP: Aspect-oriented programming)

2.6.2 Service Logic Graph (SLG)

Das zentrale Modell ist bei XMDD der Prozess. Das Prozessmodell wird dabei als so genannter „Service Logic Graph“ (SLG) dargestellt[44]. Dabei handelt es sich um einen gerichteten Graphen mit ausgezeichnetem Startknoten. Die beschrifteten Knoten symbolisieren Aktionen und beschriftete Kanten („Branches“ genannt) definieren den Kontrollfluss. Formal kann ein SLG als Kripke-Transitionssystem definiert werden. Dies ist eine Kombination aus Kripke-Struktur und beschriftetem Transitionssystem (englisch LTS, „labelled transition system“) in dem Sinne, dass sowohl die Knoten als auch die Kanten beschriftet sind. Eine formale

[44] In diesem Kontext ist mit dem Begriff „Service“ das gesamte durch den Prozess beschriebene Konstrukt gemeint, nicht eine einzelne Funktion, die von einem Servicebaustein aufgerufen wird. Ein SLG beschreibt also die Logik des gesamten Prozesses. Dieser Begriff stammt ursprünglich aus dem Telekommunikationsbereich. Details zur Geschichte sind in [MSR05] nachzulesen.

Definition ist in [Jör13] nachzulesen. Eine beschriftete Kante, also ein „Branch“ stellt die Transitionsrelation zwischen zwei Aktionen dar. Dabei ist auch der Übersichtlichkeit halber möglich (und üblich), mehrere parallele Branches (also solche mit identischem Quell- und Zielknoten) zusammenzufassen und als eine Kante mit mehreren Beschriftungen darzustellen. Jeder Branch stellt ein mögliches Ausführungsergebnis einer Aktion dar. Die Ausführungssemantik eines SLGs sieht demnach so aus, dass zunächst der Startknoten ausgeführt wird, die Ausführung dieses Knotens dann die Wahl des entsprechenden Branches bewirkt, dann der Zielknoten des Branches ausgeführt wird und so weiter. Existiert kein Nachfolgeknoten, so ist die Ausführung beendet.

Es sei hier noch angemerkt, dass ein SLG prinzipiell auch noch andere Interpretationen als die Standardinterpretation als Kontrollfluss besitzen kann. Die konkrete Semantik eines SLG wird erst durch das interpretierende Werkzeug festgelegt. In der Referenzimplementierung jABC also durch ein entsprechendes Plugin. Die mit großem Abstand geläufigste Semantik ist jedoch die Kontrollflusssemantik, also der Prozess. Dies ist auch die einzige relevante Semantik für diese Arbeit. Andere Beispiele sind zum Beispiel die graphische Modellierung von Formeln [JMS06], Datenbankschemata [Win06] oder Grammatiken [Nag09, Kapitel 4].

Ein wichtiges Feature von SLGs ist die Möglichkeit der hierarchischen Modellierung. Dabei kann ein entsprechend ausgezeichneter Knoten eine Referenz auf einen weiteren SLG symbolisieren, welcher in dem Fall als Untermodell interpretiert wird. Ein Knoten steht also für einen ganzen Prozess. Diese Hierarchien können von beliebiger Tiefe sein. Hierarchie ist wichtig, um einen SLG übersichtlich und damit verständlich zu halten. Nur so können auch komplexe Systeme modelliert werden. Außerdem entstehen so verschiedene Sichten auf das Gesamtsystem auf verschiedenen Abstraktionsleveln. Je nachdem wie detailliert und technisch die Sicht sein soll, kann man auf der obersten fachlichen Ebene stehen bleiben oder aber weit absteigen in sehr kleinschrittige, technische Abläufe. Durch die Möglichkeit, andere SLGs einzubinden, wird dabei zusätzlich die Wiederverwendung ganzer Prozesse ermöglicht. Dies wurde zum Beispiel exzessiv von Sven Jörges bei der Erstellung von Code-Generatoren getan [Jör13]. Hier wurden Code-Generatoren als SLGs beschrieben und neue Generatoren basierten fast immer auf schon bestehenden und banden diese oft auch einfach komplett oder teilweise ein.

2.6.3 Service Independent Building Block (SIB)

Die Knoten eines SLGs sind keine abstrakten Beschreibungen von Aktionen, sondern repräsentieren ausführbare Softwarebausteine, die „Service-

Independent Building Blocks“ (SIBs) genannt werden[45]. Diese repräsentieren dahinterliegende Services und sind typischerweise so grobgranular, dass sie vom Anwendungsexperten verstanden werden können.

SIBs werden als Java-Klasse realisiert. Diese enthält sowohl die Definition auch auch die Implementierung des SIBs, wobei letztere auch in eine separate Java-Klasse ausgelagert werden kann. Konkret wird ein simples POJO („Plain Old Java Object“)[46] verwendet, welches mit der Annotation `@SIBClass` versehen ist. Für die Realisierung eines SIBs in Java wird auch der Begriff „SIB-Klasse“ verwendet.

Ein SIB besteht aus folgenden Elementen:

- **UID**[47]: Eine global eindeutige Zeichenkette zur Identifizierung dieser SIB-Klasse. Diese wird direkt mit der Annotation `@SIBClass` definiert, also zum Beispiel mit einer Zeile wie dieser:

  `@SIBClass("de/tu-dortmund/sibs/example-sib")`
- **Parameter**: Eine Menge von Parametern zur Konfiguration der SIB-Benutzung. Ein Parameter hat einen Namen, einen Typen (aus einer vorgegebenen Menge von möglichen Typen) und besitzt immer einen definierten Standardwert.
- **Branches**: Eine Menge von Branches, also beschrifteten ausgehenden Kanten. Diese repräsentieren die möglichen Ausgänge einer SIB-Ausführung.
- **Dokumentation**: Dokumentierende Textbausteine für SIB allgemein, für jeden Parameter und für jeden Branch.
- **Icon**: Ein graphisches Symbol zur Visualisierung des SIBs auf der Zeichenfläche des Prozesses.
- **Label**: Eine Beschriftung des SIBs[48], die gewöhnlich unterhalb des Icons angezeigt wird.

In Listing 2.1 ist eine einfache SIB-Implementierung dargestellt, wie dies im jABC umgesetzt wird[49]. Auf das jABC als konkrete Implementierung des XMDD-Ansatzes wird in Abschnitt 2.6.4 genauer eingegangen. In der

[45] Dieser Begriff stammt (genau wie der SLG-Begriff) aus dem Telekommunikationsbereich. Hier ist also wieder mit „Service“ der gesamte Prozess gemeint [MSR05]. SIBs sind also unabhängig von dem Prozess, in dem sie benutzt werden - sie sind wiederverwendbar. Sie sind nicht unabhängig von dem Service, den sie aufrufen!

[46] Ein POJO ist eine einfache Java-Klasse, die nicht von einer bestimmten Klasse erben muss oder ein bestimmtes Interface implementieren muss.

[47] UID: kurz für „Unique Identifier“ also „eindeutiges Identifizierungszeichen“

[48] also einer konkreten SIB-Instanz in einem SLG

[49] Zu Gunsten der Übersichtlichkeit wurden die Package-Angabe sowie die Import-Anweisungen in diesem Beispiel ausgelassen.

ersten Zeile wird durch die Annotation `@SIBClass` und die damit definierte UID erreicht, dass diese Klasse als SIB interpretiert wird. In Zeile zwei erkennt man an der Implementierung der Schnittstelle `Executable`, dass dieses SIB durch den so genannten Tracer ausführbar ist. Mit der Ausführung durch Tracer beschäftigt sich Abschnitt 2.6.4.2. In den Zeilen vier und fünf ist zu erkennen, dass dieses SIB zwei feste Branches besitzt (`default` und `error`), für eine erfolgreiche Ausführung respektive eine fehlerhafte Ausführung. In Zeile sieben beginnt die Implementierung der Methode `execute`, die von der Schnittstelle `Executable` vorgeschrieben wird. In diesem Fall wird in Zeile 9 ein String aus dem Kontext gelesen (mit dem Schlüssel `x`) und in Zeile 10 ein Service aufgerufen[50], wobei der vorher aus dem Kontext gelesene String als Parameter übergeben wird. In Zeile 11 wird das Ergebnis des Service-Aufrufs unter dem Schlüssel `y` wieder in den Kontext geschrieben. Falls ein Fehler bei der Ausführung auftritt[51], wird in Zeile 14 der Branch `error` gewählt. Ist die Ausführung erfolgreich, wird dagegen in Zeile 16 der Branch `default` gewählt.

Listing 2.1: Beispielimplementierung eines einfachen SIBs

```
1  @SIBClass("ls5/test/example")
2  public class SimpleExampleSIB implements Executable {
3
4    public static final String[] BRANCHES =
5                                  {"default", "error"};
6
7    public String execute(ExecutionEnvironment env) {
8      try {
9        String data = env.get("x");
10       Object result = Service.callService(data);
11       env.put("y", result);
12     } catch (Exception e) {
13       e.printStackTrace();
14       return "error";
15     }
16     return "default";
17   }
18 }
```

Als Parameter werden alle öffentlichen[52] Felder der SIB-Klasse interpretiert. Ein Vorgabewert muss immer definiert sein. Der Wert `null` ist nicht erlaubt. Die Menge möglicher Typen enthält einfache und kom-

[50]In diesem Fall ist der Service-Aufruf ein einfacher Java-Methodenaufruf. Dahinter kann sich natürlich auch jede beliebige andere Technologie verbergen, wie zum Beispiel ein SOAP-Web-Service-Aufruf.

[51]Ein Fehler wird hier durch eine Java-Exception ausgelöst.

[52]also durch `public` gekennzeichnete

plexe Typen. Beide Kategorien enthalten sowohl Standard-Java-Typen als auch SIB-spezifische Klassen. Die einfachen Java-Typen sind zum Beispiel Boolean und String und verschiedene Zahlentypen wie Integer, Long, Float und Double. Komplexe unterstützte Java-Typen sind Listen (ArrayList, LinkedList, Vector), Mengen (HashSet, LinkedHashSet, TreeSet), Abbildungen (HashMap, TreeMap, Hashtable) oder Referenzen auf Dateien (File).

Bei den SIB-spezifischen Typen sind in dieser Arbeit besonders drei von Relevanz, da sie regelmäßig verwendet werden:

1. ContextElement
2. ContextExpression
3. ListBox

Der Parametertyp „ContextElement“ dient zur Deklaration von Kommunikation mit dem Ausführungskontext also zum Umgang mit Laufzeitdaten. Hier wird ein Schlüssel festgelegt, unter welchem im Kontext ein entsprechendes Java-Objekt zu finden ist. Durch entsprechende Annotationen an dieses Element kann optional definiert werden, ob das SIB hier ein neues Objekt in den Kontext schreibt, etwas aus dem Kontext liest, ein Objekt im Kontext manipuliert oder eben dies löscht.

Mit dem Parametertyp „ContextExpression“ können Ausdrücke in der aus dem JavaEE-Umfeld entlehnten „Expression Language“ definiert werden. Diese Ausdrücke können dynamische und konstante Bestandteile besitzen. Dynamisch kann auf Werte im Ausführungskontext zugegriffen werden oder Berechnungen können durchgeführt werden. Konstante Anteile sind zum Beispiel String-Literale oder konstante Zahlenwerte. Da hier eine technische Sprache verwendet wird, sollte genau überlegt werden, ob man ein SIB mit einem Parameter dieses Typs ausstattet. Dies führt unweigerlich dazu, dass der Benutzer dieses SIBs die Ausdruckssprache beherrschen muss. ContextExpressions sollten also im Optimalfall nur bei recht technischen SIBs verwendet werden, die auf einer eher niedrigen Stufe der Hierarchie vorkommen, so dass nur IT-Experten mit diesen SIBs in Berührung kommen. Für Business-Experten wird der Einsatz schnell zu technisch.

Der Parametertyp „ListBox“ stellt eine einfache Auswahl konstanter Werte (meist Zeichenketten) dar. In der Benutzeroberfläche zur Belegung der Parameterwerte erscheint also eine einfache Dropdown-Box.

SIBs können verschiedene Rollen ausfüllen. Die wichtigste Rolle ist wohl die des ausführbaren Bausteins. Gleichzeitig kann das SIB aber auch Daten für andere Zwecke bereit stellen, so zum Beispiel zur Überprüfung von Regeln oder Beschränkungen oder zur Code-Generierung für ver-

schiedene Plattformen. Jede Rolle wird dabei durch ein entsprechendes Interface realisiert, die von der SIB-Klasse implementiert wird.

Es hat sich mittlerweile etabliert, den Ausführungsteil eines SIBs in eine separate Java-Klasse auszulagern und in der SIB-Klasse selbst nur noch diese ausgelagerte Methode aufzurufen. Der ausgelagerte Teil wird dabei „Service-Adapter“ genannt. Die Verwendung eines separaten Service-Adapters bietet mehrere Vorteile. So ist es so möglich, verschiedene Implementierungen eines SIBs anzubieten. Außerdem ist es technisch nun möglich, ein SIB zu laden, auch wenn die Implementierung oder in der Implementierung benutzte Klassen oder Bibliotheken fehlen. Zu guter Letzt ist es bei der Code-Generierung vorteilhaft, dass nun nur noch der Service-Adapter dem Generat beigelegt werden muss. Das SIB selbst wird für die Ausführung nicht mehr benötigt. Somit hat der generierte Code keine Abhängigkeiten zum eingesetzten Framework (dem jABC, siehe Abschnitt 2.6.4) mehr.

Neben den „regulären“ SIBs, die externe Services aufrufen, existiert noch eine weitere Art von SIBs, die so genannten „Control-SIBs“. Diese SIBs dienen zur Beeinflussung des Kontrollflusses, steuern also ihre eigene Ausführungsumgebung statt externer Services. Control-SIBs können demnach mit Kontrollstrukturen von Programmiersprachen verglichen werden. Control-SIBs werden zum Beispiel für die Erstellung hierarchischer Modelle, für die Modellierung paralleler Pfade oder für die Auslösung oder die Behandlung von Ereignissen angeboten.

Das MakroSIB ist ein SIB, welches eine Referenz auf ein SLG darstellt. Dadurch werden hierarchische Modelle möglich. Das GraphSIB ist eine Variante des MakroSIBs mit dem Unterschied, dass hier ein eigener Ausführungskontext für die Ausführung des Untermodells benutzt wird. Die Kontexte sind dabei verbunden, so dass insgesamt ein hierarchischer Kontext entsteht, bei dem Unterprozesse immer auf den aktuell lokalen Kontexte sowie alle übergeordneten Kontexte zugreifen können.

Da durch MakroSIB und GraphSIB hierarchische SLGs erstellt werden, besitzen somit nicht nur SIBs Parameter und Branches, sondern auch die SLGs. Bei letzteren heißen diese dann „Modell-Parameter“ und „Modell-Branches“. Dabei werden für einen gesamten SLG globale Parameter und Branches definiert, die jeweils mit Parametern und Branches von SIBs in diesem SLG verbunden werden können. Damit werden die Werte dieser Parameter und die Ziele der Branches nicht mehr lokal im SLG sondern global für den ganzen SLG definiert. Eine Ebene höher erscheinen diese Modell-Parameter und Modell-Branches dann als Parameter und Branches des MacroSIBs bzw. GraphSIBs. Hier werden also die Werte der Modell-Parameter des referenzierten SLGs festgelegt. Modell-Branches können als „Ausgänge“ für SLGs gesehen werden. Ist ein Branch als

Modell-Branch gekennzeichnet, so wird die Ausführung bei der Wahl dieses Branches eine Ebene höher fortgesetzt und zwar durch die Wahl des entsprechenden Branches des MakroSIBs bzw. GraphSIBs.

Zur Modellierung von Parallelität werden das ForkSIB und das JoinSIB eingesetzt. Das ForkSIB startet für alle ausgehenden Branches je einen Thread. Diese Threads werden dann parallel ausgeführt. Ein JoinSIB ist das Gegenstück zum ForkSIB und wartet, bis über alle eingehenden Kanten ein Thread angekommen ist. Anschließend wird die Ausführung durch den einzigen ausgehenden Branch des JoinSIBs fortgesetzt. Ein JoinSIB synchronisiert also parallele Threads. ForkSIB und JoinSIB sind grundsätzlich paarweise und korrekt geschachtelt zu verwenden.

Zur Unterstützung von Ereignissen (oder englisch „Events“) werden ebenfalls entsprechende Control-SIBs angeboten. Das SIB FireEvent löst ein Ereignis aus, WaitForEvent wartet auf das Eintreten eines Ereignisses und AddListener registriert einen SLG als Beobachter eines Ereignisses. Nach der Ausführung dieses Control-SIBs wird also bei Eintritt des spezifizierten Ereignisses der angegebene Prozess gestartet.

2.6.4 Java Application Building Center (jABC)

Das Java Application Building Center [SMN+06, Nag09], meist abgekürzt jABC genannt, ist die Referenzimplementierung von XMDD. Es handelt sich um ein vielseitiges und erweiterbares Framework zur Entwicklung komplexer Softwaresysteme. Das jABC ist eine Java-basierte Neuentwicklung des in C++ implementierten Agent Building Centers (ABC), welches seit 1993 entwickelt wurde, und ist gleichzeitig kommerzielles Produkt sowie Experimentierplattform für die Forschung und Lehre am Lehrstuhl für Programmiersysteme der TU Dortmund.

Der wichtigste Bestandteil des jABCs ist ein Modellierungstool zur Erstellung von SLGs (siehe Abschnitt 2.6.2) aus SIBs (siehe Abschnitt 2.6.3). Durch eine modulare Plugin-Architektur ist es sehr einfach möglich, dieses Tool zu erweitern und an spezielle Anforderungen anzupassen. Eine Reihe existierender Plugins machen das jABC zu einer umfangreichen Komplettlösung für die Entwicklung prozessgesteuerter Software.

Das jABC ist ein extrem vielseitiges Tool zur Modellierung von Prozessen. Die Anwendungsgebiete umfassen dabei unter anderem wissenschaftliche Workflows, Steuerungen eingebetteter Systeme, Algorithmen des Maschinenlernens, Modellierung von Code-Generatoren oder (wie in dieser Arbeit) Business-Process-Modeling. BPM ist demnach nur ein Anwendungsgebiet unter vielen. Durch die vielen unterschiedlichen Einflüsse und historischen Wurzeln haben sich viele Ideen und Konzepte ergeben, die im BPM-Bereich bis heute immer noch wenig verbreitet

sind, dieses Gebiet jedoch deutlich nach vorne bringen können. Besonders die Untersützung der Zusammenarbeit zwischen dem Business-Experten und dem IT-Experten kann hier deutlich verbessert werden. Wie genau dies bewerkstelligt wird, wird genauer in Kapitel 5 beschrieben. Dass die Konzepte nicht auf das jABC beschränkt sind, sondern auch auf andere BPMS übertragen werden können, ist Thema von Kapitel 6.

2.6.4.1 Modellierung

Die in Abbildung 2.4 dargestellte Benutzeroberfläche des jABC ist in drei Bereiche gegliedert (siehe Nummerierung in der Abbildung):

1. **Projekt- und SIB-Browser**: Im Bereich oben links befinden sich (auf zwei Registerkarten verteilt) der Projekt-Browser sowie eine übersichtliche Darstellung der SIBs. Der Projekt-Browser (in der Abbildung nicht zu sehen) beinhaltet eine Übersicht aller jABC-Projekte, mit denen in dieser Installation gearbeitet wird. Projekte können hier zum Beispiel geöffnet, geschlossen, neu erstellt, umbenannt oder gelöscht werden. Für jedes Projekt werden die beinhalteten Elemente dargestellt. Das sind im Wesentlichen die SLGs und projektspezifische SIB-Sammlungen. Der SIB-Browser stellt in einer hierarchischen Baumansicht (Taxonomie) die zur Verfügung stehenden SIBs dar. Zu jedem SIB wird auch dessen Dokumentation samt Icon als Tooltip angeboten[53]. Wenn ein SIB nicht über die Taxonomie sondern über seinen Namen gesucht werden soll, so kann die Filter-Funktion im unteren Bereich des SIB-Browsers dazu genutzt werden. Hier kann eine Zeichenkette eingegeben werden, die Teil des SIB-Namens sein soll. Auch die Benutzung von Wildcards (*) ist hier möglich. Ist der Filter aktiv, so werden nur die SIBs angezeigt, die zum angegebenen Filter-Ausdruck passen.
2. **Zeichenfläche**: Der größte Bereich der jABC-Benutzeroberfläche ist für die Darstellung und die Bearbeitung des SLG vorgesehen. Hier können einfach per Drag-and-Drop die SIBs aus dem SIB-Browser platziert und zu Prozessen verbunden werden.
3. **Inspektoren**: Der Bereich unten links beinhaltet die so genannten „Inspektoren“, die jeweils auf einer Registerkarte dargestellt werden. Ein Inspektor ist eine Fläche in der jABC-Benutzeroberfläche, auf der Informationen zu einem bestimmten Thema dargestellt werden und in vielen Fällen auch bearbeitet werden können. Der „SIB-Inspektor“ wird zum Beispiel zur Bearbeitung der Parameterwerte

[53] Wenn der Mauszeiger für eine kurze Zeit über ein SIB im SIB-Browser positioniert wird, so wird temporär oberhalb der eigentlichen Benutzeroberfläche ein Element angezeigt, welches die SIB-Dokumentation samt Icon enthält.

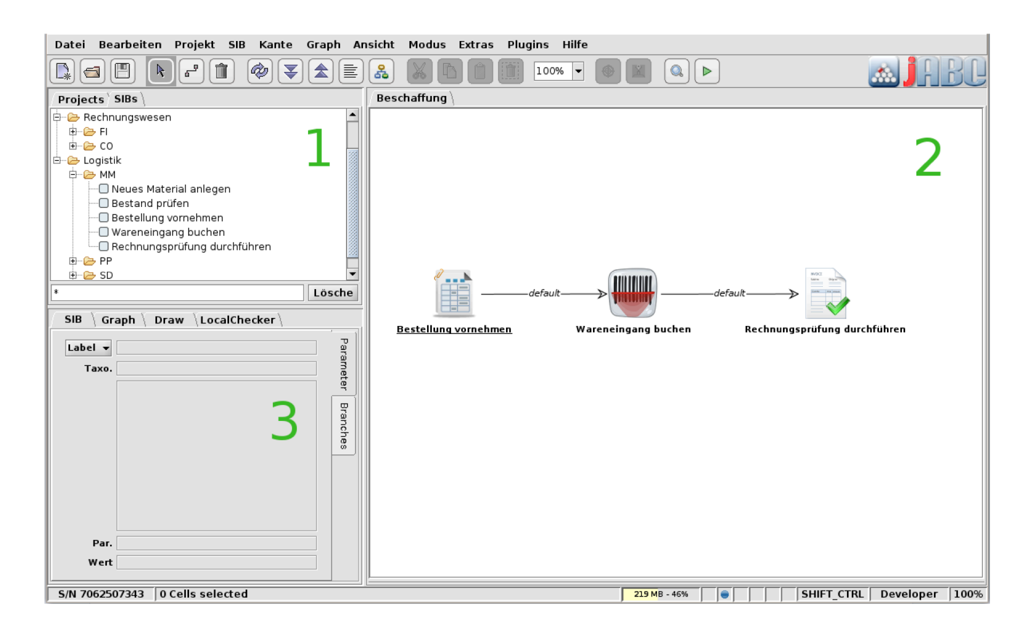

Abbildung 2.4: jABC - Benutzeroberfläche

und Branches von SIBs benutzt, der „Graph-Inspektor“ für alle Daten, die den ganzen SLG betreffen (z. B. der Name des Modells, die Modell-Parameter oder die Modell-Branches). Inspektoren können auch von Plugins genutzt werden, um eigene Informationen darzustellen. So gibt es zum Beispiel einen Inspektor für den LocalChecker, ein Plugin, welches die korrekte Benutzung der SIBs im SLG anhand lokaler Regeln überprüft.

Die Darstellung der SIBs in einer Taxonomie im Browser ist durch den Benutzer beliebig anpassbar. Zur Bearbeitung der Taxonomie existiert das Plugin „Taxonomie-Editor“. Somit ist es möglich, dass eine Sammlung von SIBs speziell für einen Benutzer angeboten wird. Genau für dessen Anforderungen können die SIBs hier in Kategorien (und Unterkategorien) einsortiert werden. Auch die Benennungen der SIBs können hier an das gewohnte Vokabular des Benutzers angepasst werden. Es existiert keine Beschränkung wie oft ein SIB in einer Taxonomie auftauchen darf. Daher ist es auch möglich, in einer Taxonomie die SIBs nach mehreren Kriterien zu sortieren. Es können also mehrere Taxonomien gleichzeitig nebeneinander in einer Taxonomie existieren.

Um die korrekte Modellierung zu vereinfachen, Fehler bei der Benutzung der SIBs also möglichst zu vermeiden, gibt es das Plugin „LocalChecker“. Hiermit können für SIBs lokale Regeln und Constraints definiert werden. Diese Regeln können zum Beispiel Wertebereiche für Parameter oder die zwingend notwendige Nutzung bestimmter Branches sein.

Bei der Bearbeitung von SLGs im jABC ist die Situation denkbar, dass die entsprechenden SIB-Klassen nicht zur Verfügung stehen, man das Modell jedoch trotzdem anschauen und bearbeiten möchte. Dies kann zum Beispiel der Fall sein, wenn nur die reine SLG-Datei ohne das umgebene Projekt (mit den projektspezifischen SIBs und Java-Bibliotheken) zur Begutachtung weiter gegeben wird. Dank eines speziellen Mechanismus, dem „Proxy-SIB", ist genau dies möglich. Beim Laden eines SLGs im jABC werden alle SIBs, deren SIB-Klasse nicht gefunden werden kann, durch ein so genanntes Proxy-SIB ersetzt. Dieses dient als Platzhalter für das eigentliche SIB. Es besitzt die gleiche Beschriftung, die gleichen Parameter (samt Werte) und die gleichen Branches wie das Original-SIB. Änderungen an Parameterwerten werden beim Speichern in die SLG-Datei übernommen. Da die Implementierung des SIBs fehlt, ist der SLG an der Stelle eines Proxy-SIBs natürlich nicht ausführbar.

Entsprechend des One-Thing-Approaches (siehe Abschnitt 2.6.1) stellt der Prozess (also der SLG) das eine zentrale Konstrukt der gesamten Systemdefinition dar. Andere Aspekte werden entsprechend an den Prozess annotiert. Auf technischer Seite existiert hierzu die Möglichkeit, beliebige Daten an einen SLG, einen SIB oder einen Branch zu annotieren. Diese Daten werden „User-Objekte" genannt. Dabei gibt es persistente User-Objekte, die mit dem SLG in der entsprechenden Datei gespeichert werden und flüchtige User-Objekte, die nur temporär für die aktuelle Sitzung gelten. Besonders die jABC-Plugins machen starken Gebrauch von dieser Technik. Beispiele für persistente User-Objekte sind atomare Propositionen für den ModelChecker GEAR (siehe Abschnitt 2.6.4.3) oder Angaben zu Benutzerrollen und -rechten. Beispiele für flüchtige User-Objekte sind die Breakpoints des Tracers (siehe Abschnitt 2.6.4.2), dessen Markierung für die aktuelle Position einer Ausführung oder verschiedenste Markierungen, die Algorithmen auf SLG an selbigen temporär hinterlassen.

Neben der Möglichkeit für die Plugins über die jABC-API Daten an SLGs anzuhängen ist es auch für den Benutzer direkt möglich, beliebige strukturierte Daten an das Modell zu annotieren. Zu diesem Zweck existiert der „ContentEditor". Dieser erlaubt es, Daten mit einer vorgegebenen Struktur an beliebige Elemente eines SLGs zu annotieren. Die Struktur wird dabei durch eine Grammatik vorgegeben, die durch ein Railroad-Diagramm definiert werden, welches selbst wiederum als SLG mit speziellen ContentEditor-SIBs erstellt wird. Der ContentEditor ist sehr vielseitig einsetzbar. Er kann unter anderem zur Vergabe von Rollen und Rechten, zur Definition von Deployment-Informationen oder zur Hinterlegung strukturierter Dokumentationstexten benutzt werden.

2.6.4.2 Ausführung

Entsprechend des agilen Ansatzes von XMDD ist die direkte Ausführbarkeit von SLGs von entscheidender Bedeutung. So bekommt der Prozessmodellierer immer direktes Feedback darüber, ob sich das Modellierte genau so verhält, wie er oder sie es sich gedacht hat. Neben der direkten Ausführung in der Modellierungsumgebung ist es auch möglich, die gleiche Ausführungsumgebung dazu zu nutzen, den Prozess ohne GUI (z. B. auf einem Server) auszuführen oder aus dem SLG ausführbaren Programmcode zu generieren, der anschließend ausgeführt (und bei Bedarf vorher kompiliert) werden kann.

Direkte Ausführung mit dem Tracer Für die direkte Ausführung des SLGs existiert der so genannte „Tracer", eine leichtgewichtige Ausführungsumgebung für SLGs. Der Tracer ist integraler Bestandteil des jABC-Frameworks. Zur Ansteuerung des Tracers existiert auch eine „Tracer-GUI", also eine graphische Benutzeroberfläche für den Tracer, als jABC-Plugin. In Abbildung 2.5 ist eine laufende Ausführung zu sehen. In einem entsprechenden Bedienfenster sind alle Steuerelemente des Tracers zugänglich. Die Ausführung kann gestartet, gestoppt und wieder aufgenommen werden. Ebenfalls ist eine schrittweise Ausführung möglich, wobei man (wie man es von Debuggern in modernen IDEs[54] kennt) wählen kann, wie man in der Ausführungshierarchie weiter navigiert. Man hat also die Wahl, ob man in ein Graph- oder Makro-SIB hinein springen und hier jeden Schritt einzeln ausführen möchte oder den ganzen SLG in einem Schritt ausführt. Es können auch Haltepunkte (engl. „Breakpoints") gesetzt werden, die den Tracer bei einer Ausführung dazu veranlassen, an diesem SIB in den schrittweisen Modus zu wechseln. Während einer schrittweisen Ausführung wird das aktuelle SIB mit einem kleinen Pfeil in grünem Kreis markiert und der zurückgelegte Weg im SLG wird durch grün eingefärbte und dicker dargestellte Kanten hervorgehoben.

Das Tracer-Fenster bietet neben der Steuerung der Ausführung auch noch Monitoring-Funktionalitäten an. Durch Betätigung des Buttons mit der Beschriftung „Details" werden verschiedene Registerkarten mit Laufzeitinformationen zur Verfügung gestellt. Hier wird unter anderem angezeigt, welche Ausführungsthreads momentan existieren[55], die Inhalte des Ausführungskontextes können eingesehen werden und die gesamte Ausführungshistorie wird in einer entsprechenden Tabelle protokolliert und dargestellt. Da das Tracer-Fenster mehrere Erweiterungspunkte besitzt, ist es für andere Plugins möglich, weitere Schaltflächen, Anzeigen

[54] Beispiele: Eclipse, Netbeans, IntelliJ Idea, Microsoft Visual Studio
[55] Die parallelen Threads können auch einzeln gesteuert werden.

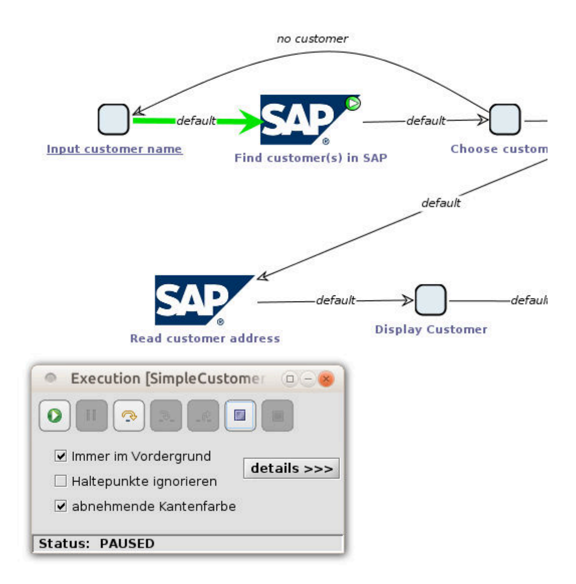

Abbildung 2.5: Ausführung im jABC mit dem Tracer

oder ganze Registerkarten einzufügen, auf denen weitere Informationen über die aktuelle Ausführung dargestellt werden.

Die Ausführung im jABC muss bei den SIBs nicht genau das gleiche Verhalten bewirken wie eine Ausführung im Produktivbetrieb. Es ist möglich, für ein SIB separate Anweisungen für beide Szenarien zu hinterlegen. Zum Beispiel kann eine Ausführung eines SIBs mit Benutzerinteraktion im jABC das Öffnen eines Dialogs[56] bewirken und im Produktivbetrieb die Navigation zu einer entsprechenden Webseite. Ein anderes Beispiel wäre die Nutzung einer Testinstallation eines ERP-Systems im jABC und der Produktivinstallation im echten Betrieb. Mit dem Tracer hat man also ein Tool zur Simulation von Geschäftsprozessen.

Code-Generierung mit Genesys Bei „Genesys“ [JMS08] handelt es sich um ein umfangreiches Framework zur Code-Generierung, also zur automatischen Erzeugung von Programmcode aus graphischen Modellen. Kern von Genesys bildet eine große Sammlung von Code-Generatoren, die aus SLGs Code in verschiedenen Programmiersprachen und für verschiedene Plattformen erzeugen. Dazu existiert auch ein entsprechendes jABC-Plugin, welches Genesys in die Benutzeroberfläche des jABC-Editors integriert. Somit kann ein SLG direkt in lauffähige Software übersetzt werden, z. B. in ein JavaSE-Programm[57], in ein Java-Servlet ein C#-Programm oder ein BPEL-Prozess. Das besondere an Genesys ist, dass die Code-Generatoren selbst wieder als SLGs beschrieben sind,

[56] zum Beispiel ein Java-Swing-Dialog
[57] ein Java-Programm, welches über seine Main-Methode gestartet wird

so dass der erste ausführbare Code-Generator durch Selbstanwendung („Bootstrapping") mit Hilfe des Tracers erzeugt werden konnte.

2.6.4.3 Model-Checking

Im Umfeld des Geschäftsprozessmanagements wird „Compliance", also der Einhaltung von Gesetzen und Richtlinien immer wichtiger. Gerade in Zeiten von Basel II bzw. Basel III oder dem Sarbanes-Oxley Act wird deutlich, dass die Zahl der zu beachtenden Vorschriften stetig wächst. Die Prozesse eines Unternehmens müssen sich sowohl entsprechend dieser allgemeinen Regeln wie auch entsprechend der Unternehmensinternen Regeln („Business-Rules") verhalten. Die Einhaltung solcher Gesetze, Richtlinien und Regeln kann mit Hilfe von „Model-Checking" überprüft werden.

„GEAR" (Game-based Easy and Reverse Model-Checking) [BMRS07, BMRS09] ist ein Model-Checker, welcher durch ein entsprechendes Plugin in das jABC eingebunden werden kann. So ist es also möglich, mathematisch zu verifizieren, dass ein SLG korrekt nach entsprechenden Regeln modelliert ist, dass also der modellierte Prozess sich so verhält, wie es in einer Menge mathematischer Regeln beschrieben wird. Diese Regeln werden, wie beim Model-Checking üblich, als temporallogische Formeln notiert. Um den Umgang mit diesen Formeln auch jemandem zu ermöglichen, der keine vertieften mathematischen Kenntnisse in diesem Bereich hat, gibt es hier Lösungen, die häufig vorkommende Patterns als fertige Eigenschaften auswählbar machen [LNMS10] oder Formeln graphisch aus vorgefertigten Blöcken zusammenbauen lassen [JMS06].

Die Grundidee der Überprüfung von Geschäftsprozessen durch Model-Checking, wie es im jABC mit GEAR durchgeführt wird, ist nicht auf das jABC beschränkt. Auch andere Prozesssprachen - wie beispielsweise BPMN - können so auf ihre Richtigkeit hin überprüft werden. In der Diplomarbeit von Jens Hornung [Hor13] wurde genau dies gezeigt. Darin wurden Prozesse BPMN 2.0 in ein Transitionssystem transformiert, welches dann wiederum durch GEAR oder alternativ durch den Model-Checker NuSMV2[58] überprüft wurde. Ein besonderer Schwerpunkt war hier die Auflösung von Parallelität, die durch die Model-Checking-Software nicht direkt unterstützt wurde.

2.6.4.4 Aufruf entfernter Tools

Mit „jETI" (Java Electronic Tool Integration) [MNS05, Kub13] existiert im XMDD-Umfeld bereits ein Tool zur Integration entfernter Ser-

[58]http://nusmv.fbk.eu/

vices. Neben einer eigenen, proprietären Remote-Execution-Technologie zur Bereitstellung von Kommandozeilenprogrammen über das Internet existieren hier auch viele Lösungen zur Einbindung von Services, die auf Standards wie WSDL/SOAP oder REST basieren. So gibt es hier zum Beispiel einen WSDL2SIB-Generator, also ein Tool, welches automatisch aus einem WSDL-Dokument ein entsprechendes SIB generiert. Dieser Technologie-zentrische Ansatz führt dazu, dass das SIB exakt so wie der Service strukturiert ist. Der in dieser Arbeit vorgestellte Ansatz (konkret in Abschnitt 5.3.5 beschrieben) beinhaltet dagegen Benutzerinteraktion zur Erzeugung der SIBs, also eine halbautomatische Generierung. Kapitel 4 enthält eine umfangreiche Analyse der unterschiedlichen Ansätze und führt Vor- und Nachteile sowie geeignete Anwendungsgebiete der Ansätze auf.

2.7 Einfachheit als Prinzip

Ein zentrales Leitprinzip bei der Entwicklung von Software im Allgemeinen und bei der praktischen Umsetzung von Geschäftsprozessen im Speziellen sollte immer die „Einfachheit“ sein. Die Betonung und Erforschung der Einfachheit ist auch Inhalt des „ITSy-Projektes“[59] [MS10, MFS11, Mar11]. Einfache Lösungen sind im Allgemeinen sehr elegant, leicht zu verstehen und leicht zu warten. Gleichzeitig könnte man einwenden, dass komplexe Probleme auch komplexe Lösungen erfordern. Dies ist jedoch nur mit Einschränkungen so zu sehen. Zunächst muss betrachtet werden, dass ein großer Teil von Problemen eben nicht komplex ist. Diesen Probleme sollten dementsprechend auch mit einfachen Lösungen begegnet werden. Nur bei wirklich komplexen Problemen sollte ein entsprechender Aufwand betrieben werden. Dabei ist immer zu bedenken, ob dieser Aufwand gerechtfertigt ist. Nach dem Paretoprinzip (auch 80:20-Prinzip genannt) werden in einem Projekt 80% der Ergebnisse in 20% der Zeit erreicht. Oft ist es also angebracht, nach 20% der Arbeit diese einzustellen und sich neuen Aufgaben zu widmen.

Ein weiterer Punkt ist, dass Einfachheit für verschiedene Personen ganz unterschiedlich wahrgenommen wird. Was für einen Kaufmann einfach ist, ist evtl. für einen Programmierer nicht einfach und umgekehrt. Daher ist es grundsätzlich wichtig, eine gute Rollenaufteilung zu finden, wo jeder seine Kernkompetenzen möglichst gut einbringen kann. Damit diese Aufteilung gut funktioniert, ist eine Etablierung einer gut funktionierenden Kommunikation zwischen den Rollen unerlässlich.

Der XMDD-Ansatz ist eine Herangehensweise, die das ITSy-Prinzip sehr

[59] ITSy: IT Simply works / It simply works

gut unterstützt. Das Prozessmodell ist sehr einfach gehalten, es gibt insgesamt nur ein Prozessmodell (der SLG) und die Rollenaufteilung ist zwischen Anwendungsexperten und IT-Experten klar geregelt.

Im Bereich der Service-Integration ist es ebenso von entscheidender Wichtigkeit, zu beachten, was für die beteiligten Personen der einfachste Weg ist. In Kapitel 4 werden verschiedene grundsätzliche Ansätze zur Service-Integration beschrieben und verglichen. Hinsichtlich ihrer Einfachheit unterscheiden sie sich immens.

Verschiedene Arbeiten wurden durchgeführt um zu demonstrieren, dass das ITSy-Prinzip auch funktioniert. In der Diplomarbeit von Jan Pardo [Par12, DGPS12] wurde ein Abrechnungssystem für Leistungen im Bereich des „Reha-Sports“ entwickelt und dabei strikt auf die Einhaltung des ITSy-Prinzips geachtet. Die Prozesse wurden hier im jABC nach dem XMDD-Ansatz modelliert. Außerdem wurde stark darauf geachtet, nie unnötig etwas selbst zu implementieren, was es bereits verfügbar als Library gibt. Wiederverwendung ist demnach ein wichtiger Baustein auf dem Weg zur Einfachheit. Teilweise konnten durch den Einsatz von Libraries komplexe Aufgabenstellungen durch sehr wenige Code-Zeilen gelöst werden.

Maik Merten zeigte am Beispiel einer Neuimplementierung des jABC-Editors als SaaS-Lösung („WebABC“), dass auch hier die Betonung und strikte Beachtung der Einfachheit zu einem sehr guten Ergebnis führt [MS12].

KAPITEL 3

Untersuchung von Business-APIs

Wenn Systeme in Prozesse integriert werden sollen, so muss es eine Möglichkeit geben, die Systeme anzusteuern. Im Bereich der ERP-Systeme haben deren Hersteller schon lange erkannt, dass es wichtig ist, hier eine Sammlung von Services anzubieten, also eine API, mit der man von außen auf die Daten und Funktionen des Systems zugreifen kann. In diesem Kapitel werden die Ergebnisse einer Untersuchung beschrieben, die die APIs verschiedener ERP-Systeme vergleicht. Die untersuchten Systeme sind dabei von sehr unterschiedlicher Natur. Vertreten sind sowohl Systeme, die von großen Konzernen eingesetzt werden als auch kleine Lösungen für kleine oder mittlere Unternehmen oder sogar Einzelpersonen. Ein weitere wichtige Kategorisierung ist die der Deployment-Art, ob es sich also um traditionelle „on-premise“-Systeme handelt, die also im unternehmenseigenen Rechenzentrum betrieben werden, oder um Cloud-basierte Systeme, bei denen das gesamte Operating an einen externen Dienstleister ausgelagert wird.

Die untersuchten APIs können als „Business-APIs“ bezeichnet werden. Diese Art von API zeichnet sich durch vier charakteristische Eigenschaften aus, die sie von anderen APIs unterscheidet:

1. Das Anwendungsgebiet ist „Business“, also die Führung eines Unternehmens und die Abläufe in einem Unternehmen (oder genauer in dessen Verwaltung) oder in einem bestimmten Markt. Die beteiligten Objekte stammen also aus dieser Domäne. Beispiele sind „Kunde“, „Mitarbeiter“, „Auftrag“ oder „Artikel“.

2. Die API ist groß. Dies bedeutet, dass die Anzahl der angebotenen Services relativ hoch ist. Da eine solche API hunderte oder gar tausende von Operationen enthalten kann, ist es unerlässlich, diese in irgendeiner Art und Weise zu organisieren.
3. Die Operationen und dazugehörigen Datentypen sind komplex. Die Operationen enthalten typischerweise eine große Anzahl an Parametern und es muss für jedes Integrationsszenario neu entschieden werden, welche Teile des Services relevant sind und welche nicht.
4. Die Operationen werden remote in einem Netzwerk angeboten. Dieses Netzwerk kann entweder ein lokales Netzwerk oder das Internet sein. Folglich kann es Clients für verschiedene Plattformen und in verschiedenen Programmiersprachen geben, weshalb Plattformunabhängigkeit hier besonders wichtig ist. Ein ebenso wichtiger Punkt ist in diesem Zusammenhang auch die Sicherheit, also z. B. eine gut funktionierende Technologie zur Authentifizierung und Autorisierung.

Die Untersuchung der Business-APIs geschieht vor dem Hintergrund des XMDD-Konzeptes. Es ist dabei entscheidend, dass die API nicht nur (wie meist üblich) von Entwicklern benutzt wird als Grundlage für eine manuelle Implementierung in einer Programmiersprache, sondern dass die API systematisch analysiert wird, auf Basis dieser Analyse ein Code-Generator entwickelt wird und dieser Generator schließlich ausführbare Prozesskomponenten (im Falle von XMDD sind dies SIBs) erzeugt, die wiederum Teil eines Geschäftsprozesses sind. Die dabei beteiligten Rollen sind in Abbildung 3.1 dargestellt. Die Gruppierung der Rollen, die durch die Swimlanes dargestellt ist, wurde von [SM07] übernommen. Demnach gibt es bei APIs drei verschiedene Stakeholder:

1. API-Designer
2. API-Benutzer
3. Benutzer resultierender Produkte

Traditionell ist ein API-Benutzer ein Entwickler[1] und der Benutzer des resultierenden Produktes ist die Person, die das Erzeugnis des API-Benutzers nutzt[2]. Der API-Designer ist das Unternehmen (oder deren Mitarbeiter), dass die API initial definiert[3] und dann bereitstellt.

Im Kontext von Business-APIs und XMDD existieren die gleichen Rollenkategorien. Es gibt jedoch mehrere API-Benutzer und mehrere Benutzer der resultierenden Produkte. Der API-Designer ist der Hersteller der

[1] z. B. einer Twitter-App für das iPhone
[2] z. B. der Nutzer der Twitter-App
[3] z. B. das Unternehmen Twitter Inc.

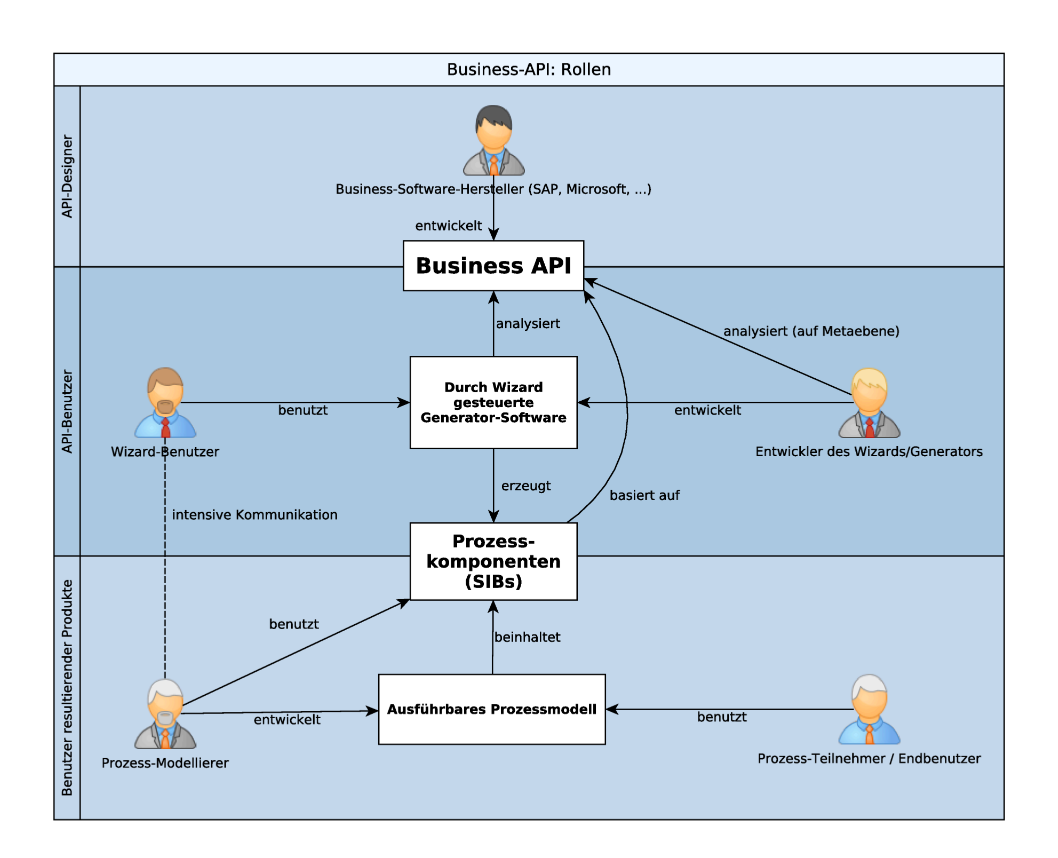

Abbildung 3.1: Rollen im Umgang mit Business-APIs

Business-Software. Dies kann also z. B. SAP oder Microsoft sein. Dieser erzeugt die Business-API, welche wiederum nicht von einem Menschen, sondern durch Software benutzt wird. Diese Software kann also quasi als ein Stakeholder gesehen werden, auch wenn es sich nicht um einen Menschen handelt. Dies impliziert, dass die API möglichst maschinenlesbar sein sollte. Die Code-Generator-Software liest und nutzt die API-Informationen. Bei InBuS (siehe Abschnitt 5.3.5) wird dieser durch einen Wizard konfiguriert. Es existieren in dieser Rollen-Kategorie also noch zwei menschliche Rollen: zum einen den Benutzer des Wizards, zum anderen den oder die Entwickler des Wizards und des Code-Generators. Die generierten Prozesskomponenten werden wiederum vom Prozessmodellierer genutzt um ausführbare Prozesse zu erstellen. Diese Person gehört demnach in die Kategorie „Benutzer resultierender Produkte", denn eine einzelne Prozesskomponente kann schon als resultierendes Produkt gesehen werden. Der entstandene Prozess ist auch wieder ein resultierendes Produkt, welches wiederum durch Personen genutzt wird.

Die Untersuchung betrachtet in Abschnitt 3.2 zunächst vier verschiedene Schnittstellen von traditionellen ERP-Systemen. Die Ergebnisse dieser Untersuchung wurden bereits in [DS11] veröffentlicht. Die betrachteten

Systeme sind dabei im einzelnen

- SAP BAPI
- SAP eSOA
- Microsoft Dynamics NAV
- Intuit Quickbooks

Intuit Quickbooks ist dabei schon eine Lösung, die es sowohl traditionell als auch als Cloud-Lösung (in dem Fall „Online-Edition“ genannt) gibt.

Hier wird diese Untersuchung nun noch um mehrere Cloud-Systeme erweitert. Dies sind im einzelnen

- Salesforce
- NetSuite
- Workday

Durch die Hinzunahme dieser Systeme kann untersucht werden, inwieweit ein Trend hinsichtlich der API-Qualität von traditionellen zu Cloud-basierten Systemen zu erkennen ist.

In Abschnitt 3.5 wird zum Schluss noch auf weitere APIs - unter anderem aus dem OpenSource-Umfeld - eingegangen.

Die Auswahl der Systeme wurde durch verschiedene Faktoren motiviert. Zunächst sollte ein möglichst breites Spektrum verschiedener Lösungen abgedeckt werden. Daher wurde darauf geachtet, Lösungen aus allen Größenordnungen zu betrachten. Außerdem sollten nur Systeme betrachtet werden, die in ihrem Marktsegment eine relevante Rolle spielen, also einen gewissen Marktanteil haben. Weiterhin spielten auch praktische Gründe, wie der mögliche Zugriff auf reale Systeme in Kooperationen mit den Herstellern oder in Industrieprojekten eine Rolle. Im Februar 2013 veröffentlichte der ESB-Hersteller MuleSoft eine Studie, die unter anderem eine Umfrage enthielt, bei der Kunden gefragt wurden, bei welchen drei bis fünf Systemen sie es für besonders wichtig ansehen, dass diese integriert werden müssen. Abbildung 3.2 zeigt einen Ausschnitt der Ergebnisse und zwar die meistgenannten Antworten. Hier ist zu erkennen, dass unter den sechs Top-Antworten fünf hier untersuchte Systeme befinden[4].

[4]Natürlich ist hier nur ein grober Trend zu erkennen. Die Antworten bezeichnen teilweise konkrete Produkte (z. B. NetSuite), teilweise Hersteller-spezifische Produktkategorien (z. B. MS Dynamics), teilweise die Hersteller selbst (z. B. SAP oder Oracle) und teilweise sehr grobe Kategorien, die nur die Funktion einer Software beschreiben (z. B. HCM, worunter auch Workday fallen würde). Somit ist nicht immer klar, was gemeint ist und der Vergleich ist manchmal schwierig. Trotzdem zeigt die Statistik einen groben Trend auf und bestätigt die Richtigkeit der Auswahl der Systeme in dieser Arbeit.

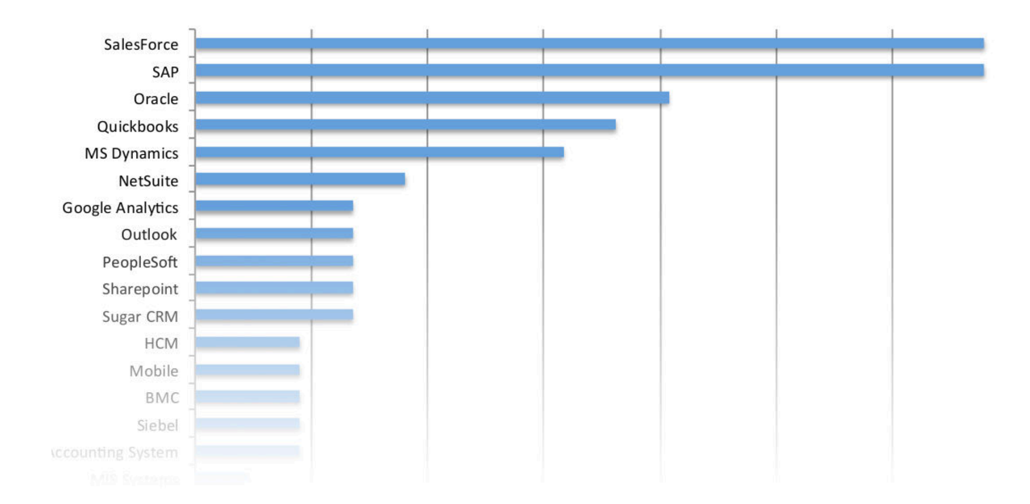

Abbildung 3.2: Top-Antworten auf die Frage „What are the top 3- 5 applications or systems your customers need to connect to?“ (Quelle: [Mul13b])

Bevor die Systeme im einzelnen untersucht werden, muss eine Reihe konkreter Anforderungen definiert werden, die eine API erfüllen sollte. Dies wird im folgenden Abschnitt 3.1 vorgenommen. Diese Anforderungen sind das Ergebnis jahrelanger Erfahrungen aus Industrieprojekten sowie studentischen Projekten im Rahmen von Projektgruppen, Seminaren, Vorlesungen und Übungen an der Technischen Universität Dortmund sowie der Universität Potsdam. Besonders die Erfahrungen aus den Diplomarbeiten von David Karla [Kar09] und André Ackermann [Ack10] waren hier besonders nützlich.

3.1 Vergleichskriterien

Wie in Abbildung 3.3 zu sehen konnten wir vier grobe Kategorien von Anforderungen an ERP-APIs identifizieren. Die erste und wichtigste Kategorie enthält alle zentralen Anforderungen. Aspekte der Benutzerfreundlichkeit wurden unter der Kategorie „API Design“ zusammengefasst. Alles, was die eingesetzte Technologie betrifft gehört zur Kategorie „Technologie“ und schließlich gehören alle Möglichkeiten, zusätzliche Informationen anzubieten in die Kategorie „zusätzliche Informationen“.

Insgesamt gibt es hier 18 verschiedene Anforderungen für ERP-APIs. Bei der Bewertung der APIs wurde grundsätzlich ein besonderer Augenmerk darauf geworfen, wie gut sich die Informationen automatisch durch Software auswerten und nutzen lassen. Dies ist besonders für die automatische Generierung von Service-Bausteinen interessant, was in Kapitel 5.3.5 dieser Arbeit detailliert behandelt wird.

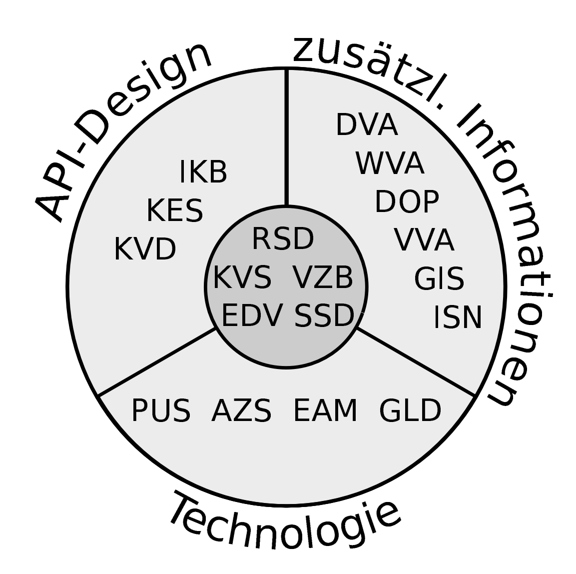

Abbildung 3.3: Kategorisierung der API-Anforderungen

Für jede Anforderung wurde ausgewertet, ob ein bestimmtes System diese Anforderung erfüllt oder nicht. Im Folgenden wird eine erfüllte Anforderung mit einem Plus-Symbol (+) bewertet, wenn sie voll erfüllt wird, mit einem Kreis (○), wenn sie zum Teil erfüllt wird und mit einem Minus-Symbol (−), falls die Anforderung absolut gar nicht erfüllt wird. Schließlich werden in zwei Übersichtstabellen die Ergebnisse gegenübergestellt. Dabei enthält Tabelle 3.7 die eher traditionellen ERP-Systeme, die bei den Unternehmen selbst in Betrieb genommen werden („on-premise“ im Englischen) und Tabelle 3.7 die Systeme, die als SaaS-Lösung „in der Cloud“ angeboten werden.

3.1.1 Zentrale Anforderungen

Zu den zentralen Anforderungen gehören diejenigen, die (eigentlich) jede ERP-API erfüllen sollte. Werden diese Anforderungen nicht erfüllt, so ist dies ein deutliches Zeichen für die allgemeine Schwäche der API.

RSD: Registry für Service-Discovery

Um sich einen Überblick über alle Services zu verschaffen, muss ein Verzeichnis (auch „Registry“ genannt) existieren, in dem alle einzelnen angebotenen Dienste aufgelistet werden. Dieses Verzeichnis kann dann zur „Service-Discovery“ genutzt werden, also zum Lokalisieren des richtigen (also des geeigneten) Services. Jeder Eintrag im Verzeichnis muss neben dem Namen mindestens noch die Information enthalten, wo der Service

deployt ist (also zum Beispiel die URL einer WSDL-Datei), damit man auf den Service zugreifen kann.

Service-Verzeichnisse können sehr unterschiedlich aussehen und technisch sehr unterschiedlich realisiert sein. Oft gibt es diese Informationen nur als reine Dokumentation, also z. B. als Webseite, als PDF oder gar als Druckwerk. Diese Dokumentation ist so konzipiert, dass sie von Programmierern als Hilfe beim Programmieren benutzt wird. Schwierig wird es, wenn eine Software das Verzeichnis automatisch auswerten soll und die Informationen richtig extrahieren soll. Bei einer gut strukturierten Webseite ist vielleicht noch mit der Technik des „Screen Scraping“ [Cha03] zu erreichen, dass man an alle benötigten Informationen gelangt. Bei dieser Technik wird versucht, gezielt Informationen aus Texten zu extrahieren, die am Bildschirm angezeigt werden. Heutzutage bezieht sich dies fast ausschließlich auf Webseiten. Ist die Seite weniger gut strukturiert, wird diese Technik jedoch immer schwieriger einzusetzen. Bei PDF-Seiten oder gar gedrucktem Text sind die Hürden entsprechend noch höher. Die automatische Auswertung ist z. B. dann wichtig, wenn die angebotenen Dienste direkt im Prozessmodellierungstool importiert werden sollen, z. B. als automatisch generierte Prozess-Bausteine (siehe Kapitel 5.3.5).

Ein weiterer Nachteil an frei formulierten Webseiten, die als Service-Verzeichnis dienen, ist, dass diese Information veralten kann. Wenn ein neuer Service hinzugefügt wird oder ein bestehender entfernt wird, so gelangt diese Information nicht automatisch in die Dokumentation. Es ist immer möglich, dass vergessen wird, den Text zu aktualisieren. Erfahrung lehrt uns, dass dies auch häufig so geschieht.

KVS: Korrekte und vollständige Service-Definitionen

Die veröffentliche Beschreibung der API entspricht leider nicht immer exakt der Implementierung. Dies ist dann der Fall, wenn die Beschreibung manuell gepflegt wird anstatt bei jeder Implementierungsänderung automatisch aktualisiert zu werden. Bei einem Ansatz, wie es zum Beispiel bei JavaDoc [Ora12] angewandt wird, kann dieses Problem nicht auftreten. Hier wird die Dokumentation der Interfaces, Klassen etc. direkt an diese annotiert um dann durch das JavaDoc-Tool die entsprechende HTML-Dokumentation automatisch zu generieren. In den Build-Tools kann nun verankert werden, dass bei jedem Kompiliervorgang automatisch auch das JavaDoc aktualisiert wird. Ähnliche Automatisierungen sollten bei allen API-Dokumentationen verwendet werden. Im Bereich der Webservices dienen meist die WSDL-Dateien selbst als ihre Dokumentation. Es existiert auch noch ein entsprechendes Dokumentations-Tag im WSDL-Standard, um jedes Element einer WSDL entsprechend zu erläutern.

VZB: Voller Zugriff auf Business-Objekte

Jede API sollte möglichst vollständig sein. Das bedeutet, dass alle Funktionalitäten, die über andere Wege (wie z.B. die graphische Benutzeroberfläche) möglich sind, auch über die API angeboten werden. Zum einen muss es zu jedem Business-Objekt (wie z.B. „Kunde“, „Rechnung“ oder „Kaufauftrag“) eine entsprechende Sammlung von Methoden geben. Diese Methoden sollten dabei mindestens die Standardoperationen zum Erzeugen, Lesen, Ändern und Löschen enthalten (englisch Create-Read-Update-Delete, kurz CRUD). Eine Leseoperation kann dabei noch dahingehend unterschieden werden, ob sie sehr einfach gehalten ist (gibt zu einer ID den dazugehörigen Datensatz zurück) oder ob es sich um eine komplexe Abfrage (englisch „Query“) handelt. Im letzteren Fall kann es zum Beispiel eine komplexe Eingabe zur Filterung der Ergebnisse geben (vergleichbar mit dem WHERE-Teil einer SQL-Abfrage) und die Ausgabe kann aus ein oder mehreren Datensätzen bestehen.

In den hier dargestellen Untersuchungen wird die Abdeckung bzgl. zehn verschiedener Business-Objekte betrachtet, die jeweils zentrale Rollen in ERP-Systemen spielen:

- Kunde (customer)
- Lieferant (vendor)
- Artikel (item)
- Angebot an Kunden (sales quote)
- Auftrag vom Kunden (sales order)
- Kundenrechnung (sales invoice)
- Angebot vom Lieferanten (purchase quote)
- Auftrag an Lieferanten / Einkaufsauftrag (purchase order)
- Rechnung vom Lieferanten (purchase invoice)
- Mitarbeiter / Angestellter (employee)

Bei zehn Business-Objekten und 5 Operationen pro Objekt ergibt sich demnach eine maximal mögliche Anzahl von 50 Operationen. Eine Abdeckung von mindestens 40 bewerten wir mit einem (+), eine Abdeckung zwischen 25 und 40 mit einem ($\circ$) und eine Abdeckung von 25 oder weniger wird mit einem ($-$) bewertet.

EDV: Eingabedatenvalidierung

Die Existenz einer Eingabedatenvalidierung stellt das wichtigste Argument für die Benutzung einer API dar im Vergleich zum direkten Zugriff auf die zugrunde liegenden Daten in der Datenbank per SQL. Ohne

Eingabedatenvalidierung wäre es unmöglich, konsistente Transaktionen zu gewährleisten und beschädigte Daten zu vermeiden. Die Validierung kann zum Beispiel einen einfachen Syntaxcheck auf einer gegebenen Zeichenkette enthalten (z.B. ob eine E-Mail-Adresse das @-Zeichen enthält) oder die Überprüfung einer komplexen Geschäftsregel (oft auch „Business-Rule“ genannt). Diese Anforderung wird im Wesentlichen von allen untersuchten Systemen zufriedenstellend und insgesamt recht ähnlich erfüllt. Daher wird dieser Punkt in den einzelnen Besprechungen der Systeme nicht weiter erwähnt.

SSD: Stabilität der Service-Definitionen

Um sicher zu gehen, dass eine Applikation, welche die ERP-API als Client nutzt, auch langfristig in der Zukunft noch funktionsfähig ist, ist es von besonderer Wichtigkeit, dass sich die Definitionen der API nicht beliebig ändern. Sobald ein Service veröffentlicht wurde, muss diese öffentliche Definition stabil bleiben, die Schnittstelle des Services darf also nicht verändert werden [Fow02]. Bei einer Änderung der Schnittstelle würden alle Clients, welche diesen Service nutzen, direkt unbrauchbar werden.

Der Grundsatz der Unveränderlichkeit von Services würde also bedeuten, dass eine Optimierung der Services unmöglich gemacht wird, was ebenfalls nicht gewünscht sein kann. Es muss möglich sein, auf sich ändernde Anforderungen zu reagieren oder auch Fehler in der existierenden Spezifikation zu beheben. Um dieses Dilemma zu beseitigen, ist die Einführung einer Versionierung der Services [Aga04, WZD07, ABP08] ein probates Mittel. Das bedeutet, dass ein Service nicht geändert wird, sondern stattdessen ein neuer, zusätzlicher Service als Alternative für den alten eingeführt wird und der alte Service als „veraltet“ (bzw. „deprecated“) markiert wird. Dieser neue Service besitzt in dem Fall meist die gleiche Bezeichnung wie sein Vorgänger, jedoch eine neue Versionsnummer. Versionierung kann dabei auf unterschiedlichen Ebenen vorgenommen werden. Es ist zum Beispiel möglich, jede Funktion einzeln zu versionieren, oder jedes Business-Objekt, oder jede Kategorie von Services oder sogar eine gesamte API.

3.1.2 API-Design

IKB: Intuitive und konsistente Benennungen

Besonders wenn APIs als Eingabedaten für automatische Generatoren dienen, ist es wichtig, dass ihre Elemente intuitive und konsistente Benennungen besitzen. Aus diesen Bezeichnern werden potentiell direkt

ganze Applikationen mit graphischen Benutzeroberflächen generiert. Sie landen also ohne Umwege beim Endnutzer. Aber auch bei der „gewöhnlichen“ Benutzung einer API durch einen Programmierer sollte dieser durch gut gewählte Namen, die sich an ein einheitliches Schema halten und nicht unnötig abgekürzt werden, so gut es geht in seiner Arbeit unterstützt werden. Permanentes Raten und Ausprobieren (Programmieren durch „Versuch und Irrtum“) sollte nicht vom Entwickler verlangt werden.

KES: Klare und einfache Struktur

Aus den gleichen Gründen wie bei der hier vor genannten Anforderung IKB ist es wichtig, dass die Struktur der Funktionen so klar und so einfach wie möglich gehalten wird.

KVD: Korrekte und vollständige Dokumentation

Eine API ist nur dann gut nutzbar, wenn sie gut dokumentiert ist und wenn diese Dokumentation möglichst vollständig ist. Nicht immer sind die Namen der Funktionen und deren Parameter selbsterklärend genug gewählt.

3.1.3 Technologie

PUS: Plattformunabhängigkeit durch den Einsatz von Standards

Es ist wichtig, dass Services nicht nur innerhalb einer gleichförmigen technischen Infrastruktur zugänglich sind. Die Beschränkung auf ein einziges Betriebssystem oder eine einzige Programmiersprache stellt einen heute kaum noch zu akzeptierenden Nachteil dar. Durch den Einsatz aktueller Standards wie z.B. SOAP Web-Services oder REST ist es möglich, komplett unabhängig von Betriebssystem oder Programmiersprache die Services anzubieten und zu nutzen.

AZS: API-Zugriffssicherheit

Wenn wichtige und sensible Daten wie Geschäftsdaten durch eine API öffentlich zugänglich gemacht werden (z.B. im ganzen Unternehmen oder sogar im Internet), ist es besonders wichtig zu beachten, dass die API ein zusätzliches Risiko für externe Attacken bedeutet. Um die Daten zu schützen muss eine sichere Authentifizierung und eine damit verbundene Autorisierung gewährleistet werden. Ebenso muss es möglich sein, die

sensiblen Daten nur verschlüsselt über das Netzwerk zu senden (z.B. mit SSL [Res01]).

EAM: Einfacher Authentifizierungsmechanismus

Die Authentifizierung muss auf der einen Seite sicher sein, sollte auf der anderen Seite jedoch keine unnötigen Hürde auf dem Weg zum Datenzugriff darstellen. Komplexe Authentifizierungsverfahren, die den Anwender immer wieder dazu zwingen die immer gleiche (manchmal recht lange) Folge von Schritten durchzuführen, ohne dass dabei die Prozedur an Sicherheit gewinnt, sind nicht wünschenswert. Im einfachsten Fall werden einfach bei jedem API-Aufruf ein Benutzername mit dem entsprechenden Passwort verlangt. Eine immer noch einfache Alternative ist ein Login-Service, welcher einen Benutzernamen und Passwort als Eingabe verlangt und bei erfolgreicher Authentifizierung einen zeitbegrenzten Schlüssel erzeugt und zurückliefert, der bei jedem weiteren API-Zugriff mit angegeben werden muss.

GLD: Geschwindigkeit (resultierend aus Latenz und Durchsatz)

Die Geschwindigkeit einer API-Technologie kann gemessen werden durch die Betrachtung der Zeit, die das System benötigt, für eine Anfrage eine entsprechende Antwort zu liefern. Diese Zeit besteht typischerweise aus einem konstanten Teil, der Latenz (die Reaktionszeit eines Funktionsaufrufs) und einem dynamischen Teil, wessen Länge von der Datenmenge abhängt („Datendurchsatz“). Die Latenz wird z.B. durch Authentifizierungs- und Autorisierungsbelange beeinflusst und der Durchsatz durch den Anteil des „Overheads“ bei der Datenkodierung. Ein hoher Overhead ist z.B. typisch für SOAP oder XML im allgemeinen. In dieser Arbeit wurde die konkrete Untersuchung (Messung) von Zeiten nicht durchgeführt, da die analysierten Systeme an sehr unterschiedlichen Orten deployt waren (lokal, gleiches Land, Übersee), so dass ein fairer Vergleich nicht möglich war.

3.1.4 Zusätzliche Informationen

DVA: Dokumentation verfügbar per API

Automatisch generierte Benutzeroberflächen sind nicht sehr gut nutzbar, wenn sie nicht ein gewisses Maß an Hilfe anbieten, z.B. in Tooltips, nach Klicken auf das obligatorische Fragezeichensymbol oder durch die Betätigung der F1-Taste. Gleiches gilt für automatisch generierte Prozessbausteine im BPMS. Auch hier erwartet der Prozessmodellierer Hilfetexte,

die ihm erklären, welcher Service durch den Prozessbaustein repräsentiert (und später aufgerufen) wird, was die Parameter bedeuten, welche Daten als Eingabe erwartet werden und was schließlich an Daten erzeugt wird. Um all dies möglich zu machen ist es für eine API nicht ausreichend, wenn die Dokumentation in einer speziellen graphischen Benutzeroberfläche oder einem Handbuch im PDF-Format vorliegt. Stattdessen muss die API selbst eine Möglichkeit anbieten, auf die passenden Hilfetexte zuzugreifen.

WVA: Wertevorschläge verfügbar per API

Generierte Benutzeroberlächen sind deutlich einfacher zu bedienen, wenn die Felder der Formulare nicht nur Freitextfelder, sondern ebenfalls einfache Auswahlfelder anbieten (z.B. „Dropdown“ oder „Combobox“ genannt). Damit so etwas möglich ist, muss die API für entsprechende Eingabefelder Wertevorschläge liefern. Diese sollten direkt „vor Ort“ verfügbar sein, dass heißt ohne den Aufruf eines weiteren Services, bei dem der Benutzer der API zunächst einmal umständlich herausfinden muss, welcher Service hier der richtige ist.

DOP: Deklaration optionaler Parameter

Es ist wichtig zu wissen, ob bestimmte Parameter einer Funktion optional oder verpflichtend sind. Ebenso können Teile komplexer Datentypen optional oder verpflichtend sein. Im Grunde bietet so gut wie jede API-Technologie irgendeinen Mechanismus an, dies zu deklarieren. Trotzdem kann es hier bei einer schlecht entworfenen API zu Problemen kommen. Es gibt zum Beispiel ein Problem, wenn mehrere Parameter als optional gekennzeichnet sind, in Wirklichkeit jedoch genau einer von beiden belegt sein muss. Entweder muss die API-Technologie hier so etwas wie eine Auswahl erlauben (was nicht bei jeder Technologie existiert) oder aus dem einen Service müssen zwei separate Services gemacht werden.

VVA: Validierungsregeln verfügbar per API

Da Validierung eine essentielle Anforderung an eine API ist, wäre es gut, wenn die entsprechenden Regeln oder Bedingungen transparent gehalten werden, so dass der Benutzer des Services die Regeln kennt, bevor der Dienst genutzt wird - und nicht erst nach vielen erfolglosen Aufrufversuchen. Diese Information würde auch dabei helfen, bessere grafische Benutzerschnittstellen zu generieren. Zum Beispiel könnte man sich ein restriktives Textfeld vorstellen, welches nur bestimmte Eingaben erlaubt

(wie zum Beispiel eine Material-ID, die immer mit einem Buchstaben beginnen muss gefolgt von einer bestimmten Anzahl an Ziffern).

GIS: Graphische Icons, die Services symbolisieren

Wenn eine API für die verschiedenen Services graphische Icons anbieten würde, so könnte man diese direkt für die Darstellung von Prozessbausteinen im Geschäftsprozessmodellierungstool verwenden. So kann der Prozessmodellierer die Bausteine einfacher erkennen, verstehen und wiederfinden. Außerdem kann durch die Benutzung von Icons der gesamte Prozess verständlicher dargestellt werden.

ISN: Internationalisierte Servicenamen

Eine API kann in verschiedenen Sprachen existieren. Dazu könnten für alle in der API benutzten Bezeichner (Namen von Business-Objekten, Parametern, etc.) Übersetzungen angeboten werden. Eine internationalisierte API ist deshalb sehr vorteilhaft, da es in dem Fall besonders für die „Business-User“ einfacher wird, die API und die daraus generierten Prozessbausteine und Benutzeroberflächen zu verstehen und zu benutzen. Nicht alle Nutzer sind zu 100% firm in der heutigen „Linua franca“ Englisch.

3.2 Traditionelle „on-premise“ ERP-Systeme

3.2.1 SAP - BAPI

SAP [SAP12b] ist ein deutscher Software-Hersteller und der weltweit größte Hersteller so genannter „Enterprise Software“ [5], also Software für die kaufmännischen Belange eines Unternehmens. Gleichzeitig ist SAP das weltweit viertgrößte Softwareunternehmen im Allgemeinen[6]. Deren Hauptprodukt ist ein ERP-System mit dem Namen SAP-ERP [SAP12a], welcher vorher als SAP-R/3 vertrieben wurde. Unter dem zuletzt genannten Namen ist das Produkt auch heute noch am bekanntesten. SAP-ERP zählt hauptsächlich mittlere bis große Unternehmen zu ihrer Zielgruppe.

BAPI [SAP05b, SAP05a] steht für *Business Application Programming Interface* und wurde Mitte der 1990er-Jahre mit SAP R/3 Release 3.1 eingeführt. Es handelt sich hierbei um eine Sammlung von Funktionen, die einen entfernten Zugriff auf das SAP-ERP-System sowie auf andere

[5] http://www.softwaretop100.org/enterprise-top-10
[6] http://softwaretop100.org/global-software-top-100-edition-2010

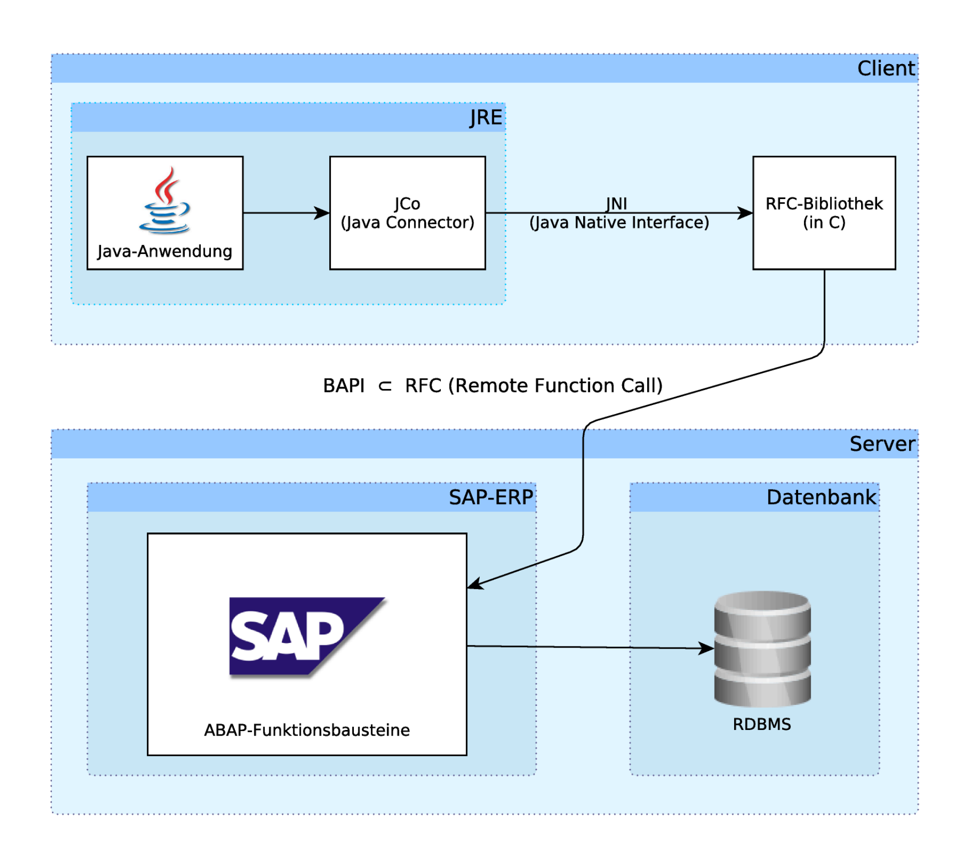

Abbildung 3.4: Zugriff einer Java-Applikation auf SAP-ERP per BAPI

SAP-Systeme erlauben. Die technologische Grundlage der BAPI ist RFC (*Remote Function Call*), ein proprietäres Protokoll von SAP, vergleichbar mit dem bekannten RPC-Mechanismus (*Remote Procedure Call*) von Sun Microsystems. Um RFCs von außerhalb eines SAP-Systems aufzurufen gibt es die RFC-Bibliothek, welche für Microsoft Windows und Linux verfügbar ist und in der Programmiersprache C verfasst wurde. Es existiert ebenfalls eine Java-Bibliothek mit dem Namen „JCo" (*Java Connector*), welche einen Zugriff auf die RFC-Bibliothek durch den Einsatz von JNI (*Java Native Interface*) ermöglicht. Die gesamte Kommunikationsstruktur von der Java-Anwendung bis zum SAP-System ist in Abbildung 3.4 dargestellt[7].

Wir testeten die BAPI-Technologie mit Hilfe einer SAP-ERP-Installation am SAP UCC („University Competence Center") der Universität Mag-

[7]Der Einfachheit halber und da es für den hier beschriebenen Aspekt keine weitere Rolle spielt, wurden die SAP-ERP-Installation und die Datenbank auf einem Server dargestellt. In der Praxis werden hierfür bei Produktivsysteme dedizierte Server eingesetzt. Bei der hier vorliegenden Betrachtung geht es jedoch lediglich um die logische Sicht von der Client-Seite aus. Dabei spielt es keine Rolle, ob sich Anwendungsserver und Datenbankserver auf einem physikalischen Server befinden.

deburg. Die Daten des Systems gehören allesamt zur fiktionalen Unternehmensgruppe „IDES“ (*International Demonstration and Education System*). Es existiert also ein vollständig installiertes und eingerichtetes SAP-ERP-System (mit vollen Customizing) für Zwecke der Evaluierung, zum Testen, sowie für Forschung und Lehre. Die angebotene Version war „SAP-ERP - version ECC 6.0“ (*ERP Central Component*).

3.2.1.1 Zentrale Anforderungen

Die Services der BAPI sind organisiert im sogenannten BOR (*Business Object Repository*), worauf man durch den BAPI-Explorer Zugriff bekommt. Der BAPI-Explorer ist Teil der SAP-GUI, der Client-Applikation für SAP, die es für verschiedene Plattformen gibt (Windows, Java, Web).

Leider kann auf das BOR nur per GUI und nicht per API zugegriffen werden. Das hat den großen Nachteil, dass man nicht per API feststellen kann, welche Services angeboten werden.

Da die Funktionen zu Business-Objekten gruppiert sind (welche selbst wieder in Kategorien zusammengefasst werden) wirkt die BAPI auf den ersten Blick als wäre sie objekt-orientiert. In Wirklichkeit handelt es sich aber nur um eine Art objekt-orientierte Sicht auf eine Sammlung von Funktionen. Jede BAPI-Funktion wird eindeutig identifiziert durch den dazugehörige Namen des RFC-Funktionsmoduls. Die Funktionen sind hier also die verwalteten Entitäten, nicht die Business-Objekte. Der Name des Funktionsmoduls entspricht dabei nicht immer dem Namen des Business-Objektes und dessen Methode bzw. der Funktionsmodulname kann nicht eindeutig aus den Namen von Business-Objekt und Methode hergeleitet werden. Zum Beispiel existiert ein Business-Objekt „SalesOrder“ mit einer Methode „ChangeFromData“. Das dazugehörige Funktionsmodul heisst dagegen `BAPI_SALESORDER_CHANGE`.

$\Rightarrow$ RSD-Bewertung: $(-)$.

Es kommt relativ häufig vor, dass Informationen im BAPI-Explorer nicht korrekt sind. Es werden zum Beispiel Parameter aufgeführt, die in Wirklichkeit gar nicht existieren oder der dokumentierende Text beschreibt eine komplett andere Funktion.

Um an die korrekten Metadaten der Funktionen zu gelangen, ist es vorzuziehen, direkt per API (also z.B. per JCo) darauf zuzugreifen. Auch per API werden alle benötigten Metadaten bereitgestellt. Diese sind (im Unterschied zum Inhalt des BOR im BAPI-Explorer) korrekt und vollständig. Die Metadaten können hier entweder direkt über bestimmte API-Funktionen abgefragt werden oder auch wahlweise aus einer von JCo automatisch generierten HTML- oder XML-Datei ausgelesen werden.

Tabelle 3.1: SAP BAPI - Business-Objekte

	Create[a]	Read		Update[b]	Delete
		Simple[c]	Query[d]		
Customer	✓	✓	–	– [d]	– [d]
Vendor	– [d]	✓	–	– [d]	– [d]
Item	–	–	–	–	–
Sales Quote[e]	✓	–	–	✓	–
Sales Order	✓	–	✓	✓	–
Sales Invoice	–	–	–	–	–
Purchase Quote	–	–	–	–	–
Purchase Order	✓	✓	–	✓	–
Purchase Invoice	–	–	–	–	–
Employee	–	–	✓	–	–

[a] hier: CreateFromData
[b] hier: Edit
[c] hier: GetDetail
[d] hier: GetList
[e] hier: CustomerQuotation

⇒ KVS-Bewertung: (○).

Die BAPI ist weit davon entfernt, komplett zu sein. Für kein einziges untersuchtes Business-Objekt sind alle notwendigen Methoden verfügbar, um das Objekt zu erzeugen, zu lesen, zu manipulieren oder zu löschen. Es können hier nur 12 von 50 möglichen Punkten vergeben werden, zu sehen in Tabelle 3.1.

Natürlich ist es möglich, die benötigten Funktionen zu implementieren und somit die BAPI zu erweitern, jedoch ist dies nicht schon fertig im Auslieferungszustand eines SAP-Systems integriert. Insbesondere bedeutet dies, dass solche Erweiterungen dann jeweils proprietär sind und nicht einheitlich für alle SAP-Systeme. Außerdem sollte man betrachten, dass jede Erweiterung Arbeit bedeutet (was Kosten verursacht) und natürlich eine potentielle Fehlerquelle darstellt.

Manche BAPI-Funktionen sind im BAPI-Explorer als „Dialog“ markiert. Das bedeutet, dass die Funktion eine graphische Benutzeroberfläche bereitstellt (in Tabelle 3.1 durch ein [d] gekennzeichnet). Das hat zur Folge, dass diese Funktion nicht von außerhalb des SAP-ERP-Systems benutzt

werden kann.

⇒ VZB-Bewertung: (−).

Die Versionierung der BAPI wird durch eines extrem simples Verfahren realisiert. Immer wenn eine neue Version einer Funktion veröffentlicht werden soll, so wird diese einfach als separate neue Funktion hinzugefügt und der Name der neuen Funktion entspricht der alten, nur dass am Ende eine Zahl angefügt wird. Bei der nächsten Version geht man dann genauso vor, nur dass die Zahl nun einfach um eins erhöht wird. Zum Beispiel wird `BAPI_CUSTOMER_GETDETAIL` ersetzt (bzw. ergänzt, da die alte Version erhalten bleibt) durch `BAPI_CUSTOMER_GETDETAIL1`, worauf dann `BAPI_CUSTOMER_GETDETAIL2` folgt. Dieses Schema wird leider nicht immer konsistent durchgehalten. Dies ist möglich, da dass ganze lediglich auf Konventionen beruht und nicht vom System verlangt wird. Zum Beispiel wurde `SalesOrder.CreateFromData` ersetzt durch `SalesOrder.CreateFromDat2` (das zweite „a“ von „Data“ fehlt nun).

⇒ SSD-Bewertung: (+).

3.2.1.2 API-Design

Die Benennungen der BAPI-Funktionen sind nicht immer konsistent, z.B. werden bestimmte Begriffe inkonsistent abgekürzt. Die „Materialart“ heißt an einer Stelle „MATERIAL_TYPE“ und an einer anderen „MATL_TYPE“. In den meisten Fällen ist die Namensgebung immerhin so intuitiv, dass man verstehen kann, was gemeint ist.

⇒ IKB-Bewertung: (◦).

Um die API möglichst einfach zu halten, ist die Anzahl der Datentypen in der BAPI sehr beschränkt. Es gibt nur einige einfache Datentypen sowie lediglich zwei Arten von komplexen Typen: *Structure* und *Table*. Eine Structure ist vergleichbar mit einem *struct* in C oder C++ oder mit einem *record* in Pascal. Es handelt sich also um einen Datensatz, welcher verschiedene Werte mit jeweils einem festen Namen und einem einfachen Datentyp besitzen. Structures können nicht verschachtelt werden, eine Structure kann also nicht eine andere enthalten. Diese Eigenschaft trifft ebenso auf eine Table zu. Diese können als Listen von Structures gesehen werden. Die hier genannten Einschränkungen führen zu einer recht aufgeräumten und übersichtlichen API, führen aber auch zu Problemen. Oft wäre es recht natürlich, tiefer geschachtelte Datentypen zu verwenden. Auch die Abbildung auf objekt-orientierte Programmiersprachen wird so erschwert.

Jede Funktion hat drei Gruppen von Parametern: Import (für die Eingabedaten), Export (für die Ausgabedaten) und Table (für Eingabe- und

```
BAPI_CUSTOMER_GETDETAIL2
  import:
   CUSTOMERNO (String)
   COMPANYCODE (optional, String)
  export:
   CUSTOMERADDRESS (Structure, 31 fields)
   CUSTOMERCOMPANYDETAIL (Structure, 15 fields)
   CUSTOMERGENERALDETAIL (Structure, 110 fields)
   RETURN (Structure, 10 fields)
  table:
   CUSTOMERBANKDETAIL (optional, 8 fields)
```

Abbildung 3.5: Struktur der BAPI zum Lesen von Kundendaten

Ausgabedaten). Die Bereiche Import und Export können Parameter enthalten, die entweder einen einfachen Datentyp haben oder eine Structure darstellen. In der Gruppe Table befinden sich nur Parameter des Typs Table. Leider kann man der API nicht direkt ansehen, ob eine Table dafür vorgesehen ist, dass man Daten in sie hineinschreibt, oder Daten aus ihr herausliest oder sogar beides. Diese Information muss der Dokumentation entnommen werden, die in natürlichsprachlichem Freitext formuliert ist. Ein Beispiel einer Schnittstellendefinition ist in Abbildung 3.5 zu sehen. Es handelt sich um eine Funktion zum Lesen von Kundendaten. Die Details zu den verwendeten Structures wurden hier ausgelassen.

⇒ KES-Bewertung: (○).

Die Dokumentation ist relativ unvollständig. Für zahlreiche Funktionen existiert gar keine Dokumentation und manchmal gibt es zwar etwas Dokumentation, aber diese gehört eigentlich zu einer anderen Funktion. Ein gutes Beispiel für so eine Fehlzuordnung zwischen Dokumentation und Implementation ist die BAPI Employee.GetList. Entsprechend der Dokumentation aus dem BAPI-Explorer ist das dazugehörige Funktionsmodul `BAPI_EMPLOYEE_GETDATA`, aber GetData sollte etwas anderes machen als GetList. In der Tat gibt es ebenfalls ein Funktionsmodul mit dem Namen `BAPI_EMPLOYEE_GETLIST` welches erfolgreich ausgeführt werden kann und die Dokumentation aus dem BAPI-Explorer passt zu diesem Funktionsmodul. Der BAPI-Explorer enthält jedoch eine Funktion mit Namen Employee.GetData. Sogar die Dokumentation, die man bekommt, wenn man per API darauf zugreift, ist entsprechend falsch (sie gehört eigentlich zu Employee.GetData). Wenn man nach der Dokumentation für Employee.GetData sucht, bekommt man kein Ergebnis.

KVD-Bewertung: (−).

3.2.1.3 Technologie

RPC ist ein proprietäres Protokoll. SAP bemüht sich jedoch, es möglich überall zu unterstützen, indem die RFC-Library für Linux und Windows angeboten wird und durch die Bereitstellung des Java Connectors (JCo).

⇒ PUS-Bewertung: (○).

Der Netzwerkverkehr von RFC-Aufrufen ist standardmäßig unverschlüsselter Klartext. Informationen wie Benutzername, Tabellennamen, Werte etc. können so sehr einfach von außen abgehört werden. Sogar das Passwort kann sehr einfach ermittelt werden, da es lediglich durch eine XOR-Operation mit einem festen und in [Nun07] veröffentlichten Schlüssel verschleiert wird. Es existiert eine SAP-Technologie mit Namen SNC (Secure Network Communications), um den Datentransfer zu verschlüsseln, was jedoch laut [Nun07] nur selten zum Einsatz kommt. Auch unser Testsystem setzte diese Technologie nicht ein.

⇒ AZS-Bewertung: (○).

⇒ EAM-Bewertung: (+).

3.2.1.4 Zusätzliche Informationen

Es gibt spezielle technische BAPI-Funktionen für den Zugriff auf die Hilfetexte (BAPI_INTERFACE_GETDOCU) und Wertevorschläge (BAPI_HELPVALUES_GET) der einzelnen Funktionen.

⇒ DVA-Bewertung: (+).

⇒ WVA-Bewertung: (+).

Die Deklaration optionaler Parameter ist nicht immer eindeutig und ausreichend. Ein gutes Beispiel ist das Hinzufügen eines neuen Kundendatensatzes. Der Parameter „personal data“ (Personendaten) ist als obligatorisch markiert, der Parameter „company data“ (Betriebsdaten) als optional. In Wirklichkeit (wie es auch in der Dokumentation der BAPI-Funktion steht) muss immer genau eines von beiden angegeben werden. Handelt es sich bei dem Kunden um eine Einzelperson, so wird „personal data“ verlangt, wenn es sich um eine Firma handelt „company data“. Wenn keines oder beides ausgefüllt wird, kommt es zu einer Fehlermeldung. Weiterhin gibt es weitere davon abhängige Parameter. Wenn „personal data“ ausgefüllt wurde, so kann in „personal address“ noch die Postadresse der Person angegeben werden, bei der Benutzung von „company data“ müssen die entsprechenden Daten in „company address“ eingegeben werden. Ein gutes API-Design hätte hier entweder so ausgesehen, dass man zwei Funktionen anbietet (eine für Personen und eine für Firmen) oder dass die API-Technologie eine Möglichkeit bietet,

```
BAPI_CUSTOMER_CREATEFROMDATA1
 import:
  PI_COPYREFERENCE (Structure, 4 fields)
  PI_PERSONALDATA (Structure, 31 fields)
  PI_COMPANYDATA (optional, Structure, 33 fields)
  PI_CONSUMEREN (optional, String)
  PI_CREDIT_CONTROL_FLAG (optional, String)
  PI_OPT_COMPANYDATA (optional, Structure, 8 fields)
  PI_OPT_PERSONALDATA (optional, Structure, 8 fields)
 export:
  CUSTOMERNO (String)
  RETURN (Structure, 10 fields)
 table: -
```

Abbildung 3.6: Struktur der BAPI zum Anlegen eines Kunden

diese komplexeren Zusammenhänge nach außen kenntlich zu machen. Die Schnittstellendefinition dieser BAPI-Funktion ist in Abbildung 3.6 zu sehen.

⇒ DOP-Bewertung: (−).

3.2.2 SAP - eSOA

Mit der Technologie „Netweaver“ hat SAP im Jahr 2004 eine neue technologische Grundlage seiner Produkte eingeführt. Es wird beworben als eine „service-orientierte Integrationsplattform, basierend auf Standards wie JEE (Java Enterprise Edition) und Web-Services“. Die proprietäre Programmiersprache ABAP wird nach wie vor unterstützt, es existiert also ein Applikationsserver mit zwei separaten Technologie-Stacks: Java und ABAP. SAP nennt seine Umsetzung einer serivce-orientierten Architektur *eSOA* (enterprise SOA) [Hai07, SAP10]. Wir testeten die eSOA-Technologie mit dem sogenannten „ES Workplace“, eine von SAP selbst bereitgestellte Webseite, die Zugriff auf verschiedene SAP-Produkte zusammen mit sehr viel Dokumentation bietet. Der ES Workplace ist hauptsächlich für Evaluierungs- und Testzwecke gedacht. Neben anderen Produkten ist dort ein vollständiges SAP-ERP (ECC 6.0) mit IDES-Daten verfügbar. Dieses System kann per SAP-GUI (die Web-Variante) oder per eSOA-Web-Services benutzt werden.

3.2.2.1 Zentrale Anforderungen

SAP bietet eine UDDI-V3-kompatible Service-Registry mit einem komfortablen Web-Interface an, womit es recht einfach gemacht wird, die richtigen Services zu finden. Auf die Registry kann also sowohl per GUI also auch per API zugegriffen werden. Zusätzlich können sämtliche UDDI-Daten als ein ZIP-Archiv heruntergeladen werden.

Tabelle 3.2: SAP eSOA - Business-Objekte

	Create	Read		Update	Delete[a]
		Simple	Query[b]		
Customer	✓	✓	✓[c]	✓	–
Vendor[d]	✓	✓	✓[e]	–	–
Item	–	–	–	–	–
Sales Quote[f]	✓	✓	✓	✓	–
Sales Order	✓	✓	✓	✓	✓
Sales Invoice[g]	✓	✓	–	–	✓
Purchase Quote	–	–	–	–	–
Purchase Order	✓	✓	✓	✓	✓[h]
Purchase Invoice[i]	✓	✓	✓[j]	–	✓
Employee[k]	✓	✓	✓	✓	✓

[a] hier: Cancel
[b] hier: Find
[c] hier: Find Customer by Name and Address
[d] hier: Supplier
[e] hier: Find Supplier by Name and Address
[f] hier: Customer Quote
[g] hier: Customer Invoice
[h] hier: Cancel Purchase Order based on Freight Order, Invoicing Preparation Cancel Request
[i] hier: Supplier Invoice
[j] hier: Find Supplier Invoice by Elements
[k] Update und Delete(Cancel) nur für Personal Address

⇒ RSD-Bewertung: (+).

Die Schnittstellen werden durch WSDL-Dateien beschrieben. Dies ist zwar eine sehr technische Sicht auf die Services, aber sie ist immer korrekt, da die aktuelle Implementierung direkt auf dieser Information basiert.

⇒ KVS-Bewertung: (+).

Die Anzahl der Services ist gewaltig. Man sollte meinen, dass alles, was man sich vorstellen kann, auch durch die API abgedeckt wird. Leider ist die Abdeckung der überprüften Business-Objekte jedoch nicht vollständig (33 von 50 Punkte) wie es in Tabelle 3.2 zu sehen ist.

⇒ VZB-Bewertung: (○).

Bei eSOA wird die Stabilität der Schnittstellen auf die gleiche Weise erreicht wie bei der oben besprochenen BAPI: Versionierung wird erreicht durch die Benutzung entsprechender Suffixe („V1", „V2", etc.) für die Service-Namen.

⇒ SSD-Bewertung: (+).

3.2.2.2 API-Design

Jeder Service in eSOA hat einen relativ intuitiven „Business-Namen" sowie eine technische Bezeichnung. Beide sind eindeutig. Zum Beispiel gibt es einen Service namens „Create Customer" mit dem technischen Bezeichner CustomerERPCreateRequestConfirmation_In. Manchmal kann es verwirrend wirken, dass das selbe Ziel über mehrere Wege erreichbar ist. Zum Beispiel gibt es einen Service zum Ändern von Kundendaten und einen zum Aktualisieren des selbigen. Mit beiden Services kann man das gleiche erreichen.

Die Benennung der Services, Operationen und Parameter sind nicht immer ganz konsistent. So gibt es zum Beispiel die Business-Objekte „customer inquiry", „customer quote" und „customer invoice", die sich alle auf den Kunden („customer") beziehen (Kundenanfrage, Kundenangebot, Kundenrechnung), der Kundenauftrag heißt hier jedoch „sales order". Lediglich in der Dokumentation des Business-Objektes „sales order" ist zu lesen, dass es sich hierbei um ein Synonym zu „customer order" handelt.

⇒ IKB-Bewertung: (○).

Die Services werden durch sehr komplexe WSDL-Dokumente beschrieben. Besonders die verwendeten Datenstrukturen weisen eine extrem hohe Komplexität auf. Es existiert genau ein WSDL-Dokument pro Service. Bei Bedarf können auch vereinfachte Sichten der WSDL-Dokumente (also bestimmte Ausschnitte daraus) abgerufen werden, z.B. nur der Service-Definition-Teil oder nur das Binding. Die folgenden Beispiele sollen verdeutlichen wie ein eSOA-Service aussieht.

Mit Hilfe der Funktion `CustomerBasicDataByIdQuery` werden Kundendaten ausgelesen. Der SOAP-Request ist in Abbildung 3.7 dargestellt. Hier kann man besonders gut erkennen, wie groß der Overhead durch die Benutzung von XML und SOAP und besonders durch die große Verschachtelungstiefe ist. Auch wenn man die API über einen automatisch generierten Java-Proxy-Stub benutzt, ist die Sicht auf die API nicht vereinfacht. Für jedes XML-Element wird eine eigene Java-Klasse erzeugt und entsprechende Objekte müssen für einen Aufruf durch ihre Konstruktoren erzeugt werden.

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:glob="...">
  <soapenv:Header/>
  <soapenv:Body>
    <glob:CustomerBasicDataByIDQuery_sync>
      <CustomerBasicDataSelectionByID>
        <CustomerID>1001</CustomerID>
      </CustomerBasicDataSelectionByID>
    </glob:CustomerBasicDataByIDQuery_sync>
  </soapenv:Body>
</soapenv:Envelope>
```

Abbildung 3.7: eSOA: SOAP-request for getting customer data

Wie man sich leicht vorstellen kann, sieht die Antwort auf diese Anfrage noch weit komplizierter aus; sie enthält sämtliche Kundendaten und das an vielen Stellen ebenso tief verschachtelt. Zum Beispiel teilt sich eine simple Angabe wie die Telefonnummer in die Teile `CountryDialingCode` und `SubscriberID` auf. Diese Elemente sind eingebettet in `Number`, was wiederum eingebettet ist in `Telephone`, was Teil von `Communication` ist, was Teil von `CommunicationData` ist, was Teil von `BasicData` ist, was letztlich eingebettet ist in `Customer`. Andere Elemente sind von ähnlicher Komplexität.

Wenn ein neuer Kunde angelegt wird, sehen die Eingabedaten überraschenderweise nicht so aus wie die Antwort auf das Lesen der Kundendaten. Zum Beispiel ist hier eine Telefonnummer sogar noch komplexer. Falls die eSOA-Schnittstellen wirklich objektorientiert organisiert wären, so würde es ein XML-Schema geben, welches die XML-Struktur der Business-Objekte definiert und die Services zum Lesen und Schreiben würden jeweils auf die gleiche Elementdefinition zurückgreifen. So könnte man zum Beispiel recht einfach einen Update-Service für Kundendaten durch die Orchestrierung der Services zum Lesen und Schreiben implementieren.

$\Rightarrow$ KES-Bewertung: ($-$).

Die Services sind insgesamt sehr ausführlich dokumentiert. Zum Beispiel gibt es ein extra Wiki mit Namen „ES Wiki“ („Enterprise Services Wiki“), welches sogenannte „ES Bundles“ enthält. Dies sind Dokumentationspakete, die sich auf eine Sammlung thematisch zusammengehöriger Services beziehen. Die Services eines ES Bundles gehören also alle zu einem Business-Prozess oder Szenario. Leider ist die Dokumentation, die man aus der Service Registry bekommt, nicht immer komplett.

$\Rightarrow$ KVD-Bewertung: ($\circ$).

3.2.2.3 Technologie

Web-Services sind eine gute Wahl, wenn großer Wert auf Plattformunabhängigkeit gelegt wird. Weiterhin nutzt SAP auch für Sicherheitsbelange Standards, und zwar HTTP basic authentication für die Authentifizierung und optional zusätzlich SSL/HTTPS zur Verschlüsselung, falls hoher Wert auf Sicherheit gelegt wird.

⇒ PUS-Bewertung: (+).

Der Sicherheits-Level kann beim Deployen eines Services konfiguriert werden. Die Authentifikation des Clients kann für einen Service komplett abgeschaltet werden oder auf HTTP basic authentication gesetzt werden oder auf die Benutzung von SSL-Zertifikaten. Das System, welches von uns evaluiert wurde, setzte HTTP basic authentication ein, was eine gebräuchliche Technologie ist, aber ohne HTTPS relativ unsicher, da Benutzername und Passwort unverschlüsselt verschickt werden (nur Base64-kodiert, was so unsicher ist wie Klartext).

⇒ AZS-Bewertung: (○).

⇒ EAM-Bewertung: (+).

3.2.2.4 Zusätzliche Informationen

Die Dokumentation der Services ist lediglich über die Webseite zugänglich. Es ist also nicht vorgesehen, diese Informationen direkt per API abzufragen.

⇒ DVA-Bewertung: (−).

Es gibt keine Unterstützung für Wertevorschläge.

⇒ WVA-Bewertung: (−).

Optionale Parameter sind als solche gekennzeichnet, ebenso optionale Teile komplexer Parameter.

⇒ DOP-Bewertung: (+).

3.2.3 Microsoft Dynamics NAV

Dynamics NAV [Mic12] ist ein ERP-System von Microsoft speziell für kleine und mittlere Unternehmen. Es wurde ehemals unter dem Namen „Navision “ von der gleichnamigen dänischen Firma entwickelt und vertrieben, welche 2002 von Microsoft übernommen wurde. In Dynamics NAV werden alle Business-Objekte durch sogenannte „Pages“ dargestellt, welche eine Menge von Standardoperationen enthalten, die sich auf

das Objekt beziehen. Die Pages sind also vergleichbar mit den Business-Objekten in SAP. Jede Page-Operation kann als Web-Service veröffentlicht werden. In der Standardeinstellung ist kein Service von außen verfügbar. Jeder Service, der veröffentlicht werden soll, muss dafür mittels der GUI von Dynamics NAV explizit in die Liste der veröffentlichen Services aufgenommen werden.

Wir testeten die NAV-Services mit einer Demo-Installation von Microsoft Dynamics NAV 2009 SP1. Diese wird ausgeliefert mit den Daten der fiktionalen Firma „CRONUS Inc.“.

3.2.3.1 Zentrale Anforderungen

Alle veröffentlichten Web-Services werden in einer zentralen Registry-Datei angeboten, die im Format „DISCO“ vorliegt (kurz für „Discovery“). Dabei handelt es sich um ein XML-basiertes Registry-Format von Microsoft [Sko02]. Diese Datei kann über HTTP abgerufen werden. Dynamics NAV nutzt das Format dabei lediglich als eine simple Liste von Service-URLs, also den URLs der WSDL-Dateien. Diese URLs enthalten immer auch den Namen der Page, so dass ein Benutzer oder Client-Programm sehen kann, welche Pages verfügbar sind und unter welcher URL sie erreichbar sind. Dass einfach alle URLs aufgelistet werden, bedeutet, dass insgesamt relativ wenige Informationen veröffentlicht werden. Für viele Nutzungsszenarios ist dies ausreichend. Besser wäre es jedoch, wenn die Page-Namen in separaten XML-Elementen stehen würden und wenn für jede Page etwas Dokumentationstext verfügbar wäre.

⇒ RSD-Bewertung: (○).

Die Schnittstellen werden durch WSDL-Dateien beschrieben. Dies ist zwar eine sehr technische Sicht auf die Services, aber sie ist immer korrekt, da die aktuelle Implementierung direkt auf dieser Information basiert.

⇒ KVS-Bewertung: (+).

Alle Pages bieten eine Menge von Standardoperationen an, welche es erlauben, das entsprechende Business-Objekt zu erzeugen, zu lesen, zu manipulieren oder zu löschen. Das führt zu einer fast vollständigen Abdeckung der Business-Objekt-Zugriffsmethoden (47 von 50 Punkte, wie in Tabelle 3.3 zu sehen).

⇒ VZB-Bewertung: (+).

Es gibt kein Versionierungskonzept, durch welches es möglich wäre, eine Stabilität der Service-Definitionen zu erreichen. Es ist jedem Benutzer (bzw. Administrator) selbst überlassen, welche Services überhaupt angeboten werden. Außerdem kann jeder Service beliebig verändert werden.

Tabelle 3.3: Dynamics NAV - Business-Objekte

	Create	Read		Update	Delete
		Simple[a]	Query[b]		
Customer	✓	✓	✓	✓	✓
Vendor	✓	✓	✓	✓	✓
Item	✓	✓	✓	✓	✓
Sales Quote	✓	✓	✓	✓	✓
Sales Order	–	✓	✓	–	–
Sales Invoice	✓	✓	✓	✓	✓
Purchase Quote	✓	✓	✓	✓	✓
Purchase Order	✓	✓	✓	✓	✓
Purchase Invoice	✓	✓	✓	✓	✓
Employee	✓	✓	✓	✓	✓

[a] hier: Read
[b] hier: ReadMultiple

Weiterhin ist es mit jeder neuen Version von Dynamics NAV möglich, dass alle Services in ihrer Standard-Konfiguration komplett anders aussehen. Hier gibt es keinerlei Gewährleistung.

⇒ SSD-Bewertung: (−).

3.2.3.2 API-Design

Die Struktur der WSDL-Dateien ist sehr klar und einfach gehalten. Zu jeder Page existiert genau eine WSDL-Datei. Die gewählten Benennungen sind dabei intuitiv und konsistent. Um zu veranschaulichen, wie die Services Dynamics NAV aufgebaut sind, folgen hier zwei Beispiele. Der Service zum Lesen von Kundendaten bekommt die Kunden-ID als Eingabe. Diese einfache Zeichenkette ist nicht in überflüssigen XML-Elementen geschachtelt. Der Rückgabewert dieses Services ist ein Element des Typen `customercard`, welcher als „complex type“ im XML-Schema der API definiert ist. Der Dienst zum Hinzufügen eines neuen Kunden ist in der gleichen WSDL-Datei definiert. Als Eingabe wird hier das gerade schon erwähnte XML-Element `customercard` erwartet. Hier kann also die Rückgabe der Lesefunktion direkt wieder als Eingabe der Schreibfunktion dienen.

⇒ IKB-Bewertung: (+).

⇒ KES-Bewertung: (+).

Leider sind die Page-Services nicht dokumentiert. Es existiert lediglich eine allgemeine Dokumentation der Standard-Operationen (create, read, update, etc.) in den Hilfetexten zu Dynamics NAV, jedoch nicht zu den Operationen bezogen auf ein bestimmtes Business-Objekt. Auch alle vorkommenden Parameter und komplexen Datentypen sind nicht dokumentiert.

⇒ KVD-Bewertung: (−).

3.2.3.3 Technologie

Die Benutzung von Web-Services verspricht im Allgemeinen eine gute Plattformunabhängigkeit. Leider nutzt Dynamics NAV proprietäre Lösungen zur Authentifizierung. Die standardmäßig verwendete Lösung ist hier Microsoft's Implementierung von SPNEGO/Kerberos. SPNEGO [rfc98] ist ein Protokoll-Verhandlungs-Mechanismus („protocol negotiation“), implementiert von Microsoft unter dem Namen „HTTP Negotiate“ [rfc06a]. Kerberos ist ein sicheres Netzwerkauthentifizierungsprotokoll, welches gegenseitige Authentifizierung („mutual authentication“) unterstützt. Das bedeutet, dass Client und Server jeweils gegenseitig ihre Identität überprüfen und verifizieren. Die hier eingesetzte Implementierung von Kerberos [rfc02, rfc06b] kann nur in einer reinen Windows-Systemlandschaft mit einem Active-Directory-Domain-Controller und einem zentralen Kerberos-Server eingesetzt werden. Somit müssen alle Benutzer, die von anderen Technologien aus (z.B. Java oder Nicht-Windows-Betriebssysteme wie Linux) auf die Services zugreifen wollen oder die eine Windows-Systemlandschaft einsetzen, die in Arbeitsgruppen („Workgroups“) anstatt Domains organisiert sind, auf die Technologie NTLM („NT Lan Manager“) ausweichen. Dies ist ebenfalls eine proprietäre Lösung von Microsoft, wird aber auch von zahlreichen Tools (z.B. wget[8], curl[9], SOAP-UI[10]) und Libraries (z.B. der HTTP-Components Client[11] aus den Apache Commons) unterstützt, die nicht für Windows entwickelt wurden.

⇒ PUS-Bewertung: (○).

Kerberos ist als sehr sichere Technologie bekannt. Das gleiche gilt nicht für NTLM. Selbst Microsoft selbst schreibt

[8] http://www.gnu.org/software/wget/manual/html_node/HTTP-Options.html
[9] http://curl.haxx.se/docs/manpage.html
[10] http://www.soapui.org
[11] http://hc.apache.org/httpcomponents-client-ga/ntlm.html

> „Therefore applications are generally advised not to use NTLM“ [12].

Es wird also empfohlen, NTLM allgemein nicht einzusetzen. Es ist generell möglich SSL/HTTPS einzusetzen, um die Sicherheit zu erhöhen.

⇒ AZS-Bewertung: (○).

⇒ EAM-Bewertung: (+).

3.2.3.4 Zusätzliche Informationen

Da die API im Allgemeinen komplett undokumentiert ist, ist demnach auch keine Dokumentation über die API abrufbar.

⇒ DVA-Bewertung: (−).

Wenn der Client zwischen verschiedenen Alternativen aus einer festen Menge von Alternativen auswählen kann, so wird in der WSDL dafür der Datentyp „enumeration“ benutzt. Diese Lösung ist noch etwas eleganter als die BAPI-Lösung, da die Information direkt in der dazugehörigen API-Funktion hinterlegt ist und kein zusätzlicher Funktionsaufruf notwendig ist.

⇒ WVA-Bewertung: (+).

Optionale Parameter sind entsprechend gekennzeichet. Dies gilt ebenfalls für optionale Teile komplexer Parameter.

⇒ DOP-Bewertung: (+).

3.2.4 Intuit Quickbooks

Quickbooks [qui12] ist eine Buchhaltungssoftware für kleine und mittlere Unternehmen vom Hersteller Intuit. Es existieren zwei Varianten dieser Software: Quickbooks Online Edition (QBOE) und Quickbooks Desktop Edition (QBDE): QBOE ist eine Webapplikation, die von Intuit selbst gehostet wird und somit dem SaaS-Paradigma folgt. QBDE ist eine traditionelle Desktop-Applikation für Microsoft Windows.

Wir testeten die Quickbooks API mit der aktuellen Version von QBOE und einer aktuellen Demoversion (11.0) von „QBDE Enterprise Solutions“.

[12] http://msdn.microsoft.com/en-us/library/cc236715(PROT.10).aspx

3.2.4.1 Zentrale Anforderungen

Quickbooks bietet seine Services nach außen an, indem ein „Posteingang“ (inbox) für Nachrichten im qbXML-Format (Quickbooks XML) angeboten wird. Dieses Format ist ein XML-basiertes Nachrichtenformat, welches durch ein entsprechendes XML-Schema definiert wird. Es gibt keine echte Registry von Services. Stattdessen kann der Client die Informationen über mögliche Services dem XML-Schema entnehmen. Zusätzlich zu dem Schema gibt es die „Onscreen Reference“, eine Webseite, die alle Elemente der API beschreibt durch das Bereitstellen textueller Dokumentation für alle Elemente. Da es sich hier um eine Webseite handelt, ist sie offensichtlich primär dazu gedacht, als graphische Benutzeroberfläche von Menschen benutzt zu werden. Trotzdem gibt es Wege, auch programmatisch an diese Informationen zu gelangen. Dies ist recht einfach möglich, da die Onscreen Reference alle Informationen aus einer Menge von JSON-Dateien (JavaScript Object Notation) bezieht, welche auch mit dem Quickbooks SDK ausgeliefert werden, einer Sammlung von Tools, mit denen man Quickbooks-bezogene Software entwickeln kann. Bei der Entwicklung von Windows-Clients hat man die zusätzlichen Möglichkeit, die QBFC (Quickbooks Foundation Classes) zu benutzen, einer Software-Bibliothek, die eine API basierend auf Microsoft's COM-Technologie[13] enthält. Diese versteckt die Benutzung von XML vor dem Benutzer.

⇒ RSD-Bewertung: (∘).

Es stellte sich heraus, dass die Informationen in dem XML-Schema nicht immer korrekt und vollständig sind. Die JSON-Dateien sind hier von deutlich besserer Qualität. In [Ack10] wird das Schema durch die systematische Analyse der JSON-Files nachträglich korrigiert und erst anschließend als Grundlage für die automatische Generierung von Prozessbausteinen genutzt.

⇒ KVS-Bewertung: (∘).

Die Abdeckung der Business-Objekte ist leider fast so schlecht, wie bei der SAP-BAPI (15 von 50 Punkte, wie in Tabelle 3.4 zu sehen).

⇒ VZB-Bewertung: (−).

Es existiert kein Konzept zur Versionierung der API, wodurch es möglich wäre eine Stabilität derselbigen zu gewährleisten.

⇒ SSD-Bewertung: (−).

[13]COM: „Component Object Model“

Tabelle 3.4: Quickbooks - Business-Objekte

	Create	Read		Update	Delete
		Simple	Query		
Customer	✓	–	✓	✓	✓[a]
Vendor	✓	–	✓	✓	–
Item	–	–	✓	–	–
Sales Quote	–	–	–	–	–
Sales Order	–	–	–	–	–
Sales Invoice[b]	✓	–	✓	–	–
Purchase Quote	–	–	–	–	–
Purchase Order	–	–	–	–	–
Purchase Invoice[c]	✓	–	✓	–	–
Employee	✓	–	✓	✓	–

[a] kein Löschen, sondern per Update isActive auf „false“ setzen
[b] hier: Invoice
[c] hier: Bill

3.2.4.2 API-Design

Die Namen für Business-Objekte, Operationen und Parameter sind intuitiv verständlich und konsistent gewählt.

⇒ IKB-Bewertung: (+).

Die Struktur der qbXML-Dokumente ist meistens klar und einfach gehalten. Für jeden Service ist eine Anfragenachricht („Request“) sowie eine dazugehörige Antwortnachricht („Response“) definiert. Zum Beispiel existiert zum Hinzufügen eines neuen Kundendatensatzes das Element `<CustomerAddRq>` als Request und `<CustomerAddRs>` als Response. Der Request enthält das Element `<CustomerAdd>` welches wiederum alle Elemente eines Kunden enthält, was z.B. der Name, die Adresse und Kommunikationsdaten wären. Der Response enthält das Element `<CustomerRet>`, welches ebenfalls die neu angelegten Kundendaten enthält und zusätzlich die automatisch erzeugte ID des Kunden.

⇒ KES-Bewertung: (+).

Die Dokumentation ist relativ detailliert aber leider nicht an allen Stellen komplett.

⇒ KVD-Bewertung: (○).

3.2.4.3 Technologie

Die Art und Weise wie qbXML-Dokumente an das System übertragen werden, hängt von der eingesetzten Quickbooks-Variante (QBOE oder QBDE) ab.

Online Edition:

Die Kommunikation mit Quickbooks Online Edition wird realisiert durch das Senden von qbXML-Dokumenten über HTTPS. Dies ist auf der einen Seite eine plattformunabhängige und sichere Lösung aber auf der anderen Seite recht umständlich, da für jeden Funktionsaufruf wohlgeformte XML-Dokumente erzeugt werden müssen anstatt einfach eine API-Funktion aufzurufen.

⇒ PUS-Bewertung: (○).

⇒ AZS-Bewertung: (+).

Die Authentifizierung gegenüber QBOE ist äußerst umständlich. Zuerst muss man sich für das „Intuit Developer Network“ registrieren um eine so genannte *AppID* sowie einen *AppLogin* zu bekommen. Mit diesen Daten kann sich die Client-Applikation, die mit dem SaaS-Angebot

kommunizieren will, identifizieren. Wenn der erste Verbindungsaufbau zu QBOE aufgebaut werden konnte, muss ein *ConnectionTicket* erzeugt werden, welches diese konkrete Instanz der Applikation identifiziert. Um ein ConnectionTicket zu erzeugen, muss der Benutzer seinen Benutzernamen und sein Passwort für die Quickbooks-Webseite angeben. Dies muss für jede Sitzung erneut geschehen. So wird ein *SessionTicket* erzeugt, welches den Zugriff auf QBOE ermöglicht und automatisch ausläuft, wenn über mehr als eine Stunde keine Aktivität festgestellt wird.

⇒ EAM-Bewertung: (−).

Desktop Edition:

Auf die Desktop-Edition von Quickbooks kann nur von einem Windows-System aus zugegriffen werden, da die Kommunikation auf Microsoft's COM-Technologie basiert. Wenn auf QBDE von anderen Systemen aus zugegriffen werden soll, muss eine Art „Wrapper“ implementiert und auf dem Windows-System installiert werden, welcher selbst per COM auf QBDE zugreift und den Service nach außen über eine plattformunabhängige Technologie wie z.B. Web-Services oder direkt über einfache TCP/IP-Sockets wie in [Ack10] anbietet.

⇒ PUS-Bewertung: (−).

Da COM für die Kommunikation verwendet wird, wird die Authentifizierung über die Standard-Windows-Authentifizierung (NTLM) durchgeführt. So existieren hier also die gleichen Sicherheitsbedenken wie bei Dynamics NAV.

⇒ AAS-Bewertung: (○).

Wenn eine Verbindung mit QBDE aufgebaut wird, erscheint ein Autorisierungsdialog, der nach dem Benutzernamen sowie dem Passwort fragt.

⇒ EAM-Bewertung: (○).

3.2.4.4 Zusätzliche Informationen

Die Dokumentation ist nicht dafür gedacht, dass auf sie per API zugegriffen wird. Trotzdem ist es möglich aufgrund der Tatsache, dass die JSON-Dokumente einen Link auf die textuelle Dokumentation enthalten.

⇒ DVA-Bewertung: (○).

Genau wie Dynamics NAV benutzt auch Quickbooks den XSD-Datentyp `enumeration` um eine Auswahl aus verschiedenen Alternativen anzubieten. Wenn zum Beispiel jemand einen bestimmten Kunden finden möch-

ten, so gibt es den Service `CustomerQuery`, welcher unter anderem einen `NameFilter` erwartet, der wiederum aus `MatchCriterion` und `Name` besteht. Dabei ist `MatchCriterion` eine `enumeration` mit den möglichen Werten `StartsWith`, `Contains` und `EndsWith`. *Name* ist ein Freitextfeld.

⇒ WVA-Bewertung: (+).

Optionale Parameter werden als solche gekennzeichnet. Gleiches trifft auf optionale Teile komplexer Parameter zu. Außerdem ist klar definiert, wenn die Eingabe ein Element aus einer Liste von Alternativen enthalten muss.

⇒ DOP-Bewertung: (+).

3.3 Cloud-basierte ERP-Systeme

Wie in Abschnitt 2.4 beschrieben, handelt es sich bei „Cloud-Computing“ um ein aktuelles Hype-Thema, dem sich auch die ERP-Hersteller nicht entziehen können und wollen. Die Vorstellung, ein fertiges, direkt zu benutzendes ERP-System „in der Cloud“ zur Verfügung zu haben, scheint zunächst einmal sehr verlockend zu sein. So besteht die Chance, sich weder um die Aspekte Skalierbarkeit oder Sicherheit, noch um die Administration im Allgemeinen kümmern zu müssen. Dies spart Personal, Arbeitszeit und somit Kosten.

Es sind verschiedenste Modelle vorstellbar, ein ERP-System in der Cloud anzubieten, jeweils mit unterschiedlich gelagerten Vor- und Nachteilen. Eine offensichtliche Lösung ist ein ERP-System als SaaS. Dies ist für existierende Produkte wie beispielsweise ein SAP-ERP schwierig zu realisieren. Hier bietet sich eher ein Deployment des Produktes auf eine IaaS an. Kiadehi und Mohammadi [KM12] vergleichen diese Ansätze und analysieren die jeweiligen Vor- und Nachteile.

SAP ist zwar kein Vorreiter im Cloud-Computing-Umfeld, hat jedoch die hier liegenden enormen Möglichkeiten erkannt und dementsprechend viel in diese Richtung investiert. Mittlerweile ist „Cloud“ einer der Eckpfeiler der aktuellen SAP-Strategie. Dabei sind sehr verschiedene Produkte entstanden. Das erste auch so beworbene „Cloud-ERP“ von SAP ist die SaaS-Lösung „Business ByDesign“ [SAP13a]. Dies wurde 2007 veröffentlicht und zielt auf kleine und mittlere Unternehmen (KMUs) mit mindestens 10 Personen ab. Die Benutzeroberfläche ist web-basiert und wurde mit Microsoft’s Silverlight-Technologie entwickelt. Dies bedeutet, dass die Applikation praktisch nur von Clients mit Microsoft Windows benutzt werden kann. Es existiert zwar auch eine Linux-Implementierung von Silverlight namens „Moonlight“, dessen Entwicklung jedoch mittler-

weile schon wieder eingestellt wurde[14]. Außerdem werden nur der Internet Explorer sowie Firefox als Browser unterstützt. Die fehlende Plattformunabhängigkeit verstößt dabei gegen die grundlegenden Prinzipien des Cloud-Computings. Business ByDesign basiert wie SAP-ERP auf der Netweaver-Technologie und eSOA. Das bedeutet, das für die API die gleiche Technologie eingesetzt wird wie beim traditionellen SAP-ERP. Daher wurde diese Lösung hier nicht separat untersucht.

Eine weitere SAP-Lösung nennt sich „SAP Business One“. Diese sehr einfache ERP-Software für kleine Unternehmen (schon ab einem Mitarbeiter), wird seit kurzem auch unter der Bezeichnung „SAP Business One OnDemand“ als Cloud-Lösung angeboten [SAP13b]. Das bedeutet in diesem Fall, dass die Software einfach über einen ActiveX-basierten Remote-Desktop angeboten wird. Die Installation befindet sich auf einem Windows-Server, auf den man per Remote-Desktop zugreifen kann, der wiederum im Browser dargestellt wird. Es ist geplant, die Benutzeroberfläche Schritt für Schritt in eine Web-basierte Lösung zu migrieren. Diese Lösung funktioniert also ebenfalls lediglich unter Microsoft Windows. Business One wurde bereits 2002 veröffentlicht, zwei Jahre vor Netweaver und eSOA, basiert also nicht auf diesen Technologien. Für Integrationszwecke gibt es eine so genannte „DI-API“ („data integration“), deren Implementierung als Windows-DLL[15] vorliegt und auf Microsoft's COM-Technologie basiert. Für Zwecke, bei denen von Nicht-Windows-Systemen auf die Services zugegriffen werden muss, existiert zusätzlich ein Web-Service-Wrapper namens „B1WS“[16]. Diese Software ist unter der SDN-Lizenz[17] veröffentlicht, was bedeutet, dass der Quelltext öffentlich zugänglich ist, es jedoch keinen Support und keine Garantien jeglicher Art gibt. Dieser sehr inoffizielle Charakter der Web-Service-API ist der Grund, warum diese Technologie hier nicht weiter betrachtet wurde. Nichtsdestotrotz ist es interessant zu sehen, dass sich diese Technologie grundlegend von SAP's eSOA unterscheidet. Eher wie bei Dynamics NAV von Microsoft wird hier ein WSDL-Dokument pro Business-Objekt verwendet und jedes Dokument beinhaltet dabei die gleichen einfachen CRUD-Operationen.

Auch SAP's Hauptprodukt SAP-ERP wird als Cloud-Lösung angeboten. Dazu kooperiert SAP mit dem IaaS-Anbieter Amazon und bietet SAP-ERP-Installationen auf Amazon's EC2[18] an[19]. Hierzu werden von Amazon Anleitungen und allgemeine Dokumentationen dazu angeboten,

[14]http://www.infoq.com/news/2012/05/Miguel-Moonlight
[15]DLL: „Dynamic Link Library
[16]B1WS: „Business One Web Services“
[17]SDN: „SAP Developer Network“
[18]EC2: „Elastic Compute Cloud“
[19]http://aws.amazon.com/de/sap/

wie ein SAP-ERP auf EC2 deployt und betrieben werden kann und was dabei beachtet werden muss. SAP und Amazon zertifizieren dabei, dass diese und andere SAP-Lösungen auf dieser Plattform lauffähig sind. Da hier ein Standard-SAP-ERP zum Einsatz kommt, muss hier keine separate Evaluierung der API vorgenommen werden.

Microsoft bietet Dynamics NAV ebenfalls als Cloud-Installation an. Dafür wird die IaaS-Lösung „Windows Azure“ eingesetzt, wobei hierauf der Server von Dynamics NAV mit einer Web-Oberfläche deployt wird. Dieser Dienst wird seit dem vierten Quartal 2012 angeboten. Da die Art des Deployments (Cloud oder traditionell im eigenen Rechenzentrum) nichts an der API des Systems ändert, war es nicht nötig, hier eine separate Evaluation der API vorzunehmen.

Quickbooks hat mit seiner „Online Edition“ seit jeher eine Cloud-Lösung im Portfolio. Dieses wurde bereits in Abschnitt 3.2.4 behandelt. Die API unterscheidet sich im Wesentlichen nicht von der Desktop-Variante. Lediglich im Bereich der technologischen Aspekte gibt es Unterschiede, da die Authentifizierung anders erfolgt.

Im Folgenden werden drei APIs von Systemen untersucht, die man als echte, „native“ Cloud-Systeme bezeichnen kann. Dabei handelt es sich um Lösungen, die alle ursprünglich direkt für die Cloud entwickelt wurden. Es ist von besonderem Interesse, zu untersuchen, inwieweit die APIs dieser Lösungen qualitativ besser sind als die APIs der traditionellen Lösungen. Die untersuchten Systeme sind Salesforce (Abschnitt 3.3.1), NetSuite (Abschnitt 3.3.2) sowie Workday (Abschnitt 3.3.3).

3.3.1 Salesforce

Salesforce [sal12a] ist ein CRM-System („Customer Relationship Management“), welches als SaaS angeboten wird. In einer SaaS-Lösung ist das Web-Frontend im besten Falle nicht die einzige Schnittstelle. Zusätzlich gibt es fast immer auch eine entsprechende API, so dass andere Softwareprodukte mit diesem System interagieren können. Laut Salesforce selbst entstanden schon 2008 57% von deren Datenverkehr durch API-Aufrufe [Sal08]. Besonders durch das starke Wachstum im Bereich der Smartphone-Apps ist hier zu erwarten, dass der Anteil der API-Aufrufe seitdem noch dramatisch angestiegen ist und auch noch weiter wachsen wird.

Von Salesforce werden verschiedene Arten von APIs angeboten. Zunächst existiert eine SOAP/WSDL-Web-Service-API. Zusätzlich gibt es eine REST-API, die dieselbe Funktionalität anbietet und eine REST-basierte so genannte Bulk-API speziell für große Datenmengen und eine SOAP/WSDL-basierte Metadaten-API, die Zugriff auf Daten erlaubt,

die das Customizing des Angebots betreffen. In dieser Arbeit liegt der Fokus hauptsächlich auf der SOAP/WSDL-Web-Service-API, da sich diese am besten mit den anderen hier untersuchten APIs vergleichen lässt. Aus der „Business-Sicht“ ist die Rest-API mit der hier untersuchten identisch, da hier exakt die gleichen Funktionen und Daten angeboten werden. Der Abschnitt über Technologiebelange (Abschnitt 3.3.1.3) wird kurz einige Aspekte der REST-API ansprechen.

Um es einfacher zu machen, die API in Java zu nutzen gibt es ein spezielles Web-Service-Framework, welches von Salesforce selbst entwickelt wird mit Namen „WSC“ („Web Service Connector“). Dieses Framework enthält ein Tool, welches aus den Salesforce-WSDLs Java-Stubs generiert sowie eine Laufzeitbibliothek für die Java-Software, von der aus die Services aufgerufen werden sollen.

Dem Client werden zwei verschiedene Arten von WSDL angeboten. Die erste ist die „Enterprise WSDL“. Diese bietet stark getypte Schnittstellen mit allen Betriebsdaten. Die WSDL kann zum Beispiel durch die hauseigene IT-Abteilung genutzt werden, um spezielle Client-Applikationen für das Unternehmen selbst zu entwickeln. Da dies genau das ist, was in dieser Arbeit unterstützt werden soll, wurde genau diese API hier untersucht. Die zweite WSDL ist die allgemeiner gehaltene so genannte „Partner WSDL“, die schwach typisiert ist und genutzt werden kann von Software-Unternehmen, die allgemeine Clients (zum Beispiel Smartphone-Apps) entwickeln wollen, durch die man Zugriff auf beliebige Kundeninstanzen von Salesforce bekommt.

3.3.1.1 Zentrale Anforderungen

Die SOAP/WSDL-basierte Web-Service-API bietet insgesamt nur ein WSDL-Dokument (die „Enterprise WSDL“) für die Gesamtheit aller Services an. Da dieses Dokument die komplette API enthält, ist sie naturgemäß extrem groß (7642 Zeilen, ohne benuterspezifische Services, die während des Customizings hinzugefügt werden können). Dieser Ansatz funktioniert für Salesforce, da der Gesamtumfang des Produktes noch relativ überschaubar ist im Vergleich zu zum Beispiel SAP. Aufgrund der Tatsache, dass es sich „nur“ um eine CRM-Software handelt und sich daher auf den einen Aspekt der Kundenverwaltung konzentriert, ist diese eine WSDL immer noch zu handhaben. Die Benutzung nur eines WSDL-Dokuments macht es möglich, ohne eine Registry auszukommen, da der Client nie in die Situation gerät, das „richtige“ WSDL-Dokument aus einer großen Menge zu identifizieren.

Natürlich existiert weiterhin die Notwendigkeit, innerhalb der WSDL die richtigen Services zu finden. Dazu gibt es mehrere Operationen, deren

Namen jeweils mit `describe` beginnen. Diese bieten Zugriff auf die Metadaten der Services. Zum Beispiel liefert `describeGlobal` eine Liste aller verfügbaren Business-Objekte und `describeSObject` gibt die Informationen zu einem speziellen Business-Objekt zurück (z.B. welche Felder es beinhaltet).

⇒ RSD-Bewertung: (+).

Da WSDL-Dokumente und die automatische Generierung von Java-Stubs aus selbigen eingesetzt werden, besteht nicht die Gefahr, dass die API-Beschreibung nicht aktuell ist. Sie ist natürlicher Bestandteil der Implementation.

⇒ KVS-Bewertung: (+).

Auf alle Business-Objekte der Salesforce-Software kann per API zugegriffen werden. Es existiert nichts in der GUI, was nicht auch von der WSDL angeboten wird. Natürlich muss auch hier wieder beachtet werden, dass dies für Salesforce einfacher ist als für eine deutlich umfangreichere Software-Suite wie SAP-ERP. Auf jedes Objekt kann über jede CRUD-Operation zugegriffen werden (bei entsprechenden Rechten des Clients). Für komplexe Abfragen wird die proprietäre Abfragesprache SOQL („Salesforce Object Query Language“) eingesetzt, wobei es sich um eine vereinfachte Variante von SQL handelt. Die SOQL-Artefakte müssen mittels des Web-Services `query` an eine Salesforce-Instanz geschickt werden. Wenn der Client nach einem bestimmten Objekt suchen möchte, so gibt es hierfür wieder eine eigene proprietäre Sprache: SOSL („Salesforce Object Search Language“). Die entsprechenden Artefakte werden dabei per Web-Service `find` an Salesforce geschickt.

Da Salesforce kein vollwertiges ERP-System, sondern „nur“ ein CRM-System ist, wäre es nicht wirklich fair, die gleiche Menge an Business-Objekten zur Untersuchung von deren Unterstützung heranzuziehen.

⇒ VZB-Bewertung: (+).

Anstatt dass einzelne Services versioniert sind, hat hier die gesamte API eine einzige Versionsnummer. Es gibt dabei keine Garantie, dass ein Client, der gegen eine bestimmte API-Version implementiert wurde, auch mit einer zukünftigen Version funktionieren wird. Es ist möglich auf ein neues Salesforce-Release zu migrieren und dabei bei der alten API-Version zu bleiben. Es gibt grundsätzlich drei Jahre Support für eine bestimmte API-Version. Nach Ablauf der Garantiezeit gibt es keinerlei Gewährleistungen mehr.

⇒ SSD-Bewertung: (○).

3.3.1.2 API-Design

Die Benennungen der API sind intuitiv und konsistent.

⇒ IKB-Bewertung: (+).

Die Struktur ist klar und einfach.

⇒ KES-Bewertung: (+).

Die API-Technologie ist sehr gut dokumentiert. Auch alle Business-Objekte und deren Felder sind ausführlich auf der Salesforce-Webseite [Sal12b] dokumentiert.

⇒ KVD-Bewertung: (+).

3.3.1.3 Technologie

Mit WSDL, SOAP und REST werden plattformunabhängige Standards eingesetzt wodurch die API von beliebigen Plattformen und Programmiersprachen aus benutzt werden kann.

⇒ PUS-Bewertung: (+).

Der Datenverkehr wird bei Salesforce grundsätzlich verschlüsselt. Dies wird durch die strikte Verwendung von HTTPS (SSLv3, TLS) erreicht sowie der Regel, dass die Schlüssellänge mindestens 128 Bit betragen muss. Zusätzlich kann ein Benutzer nur dann auf ein Salesforce-System zugreifen, wenn für ihn das Zugriffsrecht „API enabled“ aktiviert ist. Der Client muss immer zunächst ein Login durchführen (mittels eines entsprechenden Web-Services) und erhält dadurch ein so genanntes *Security Token*, wodurch man Zugriff auf das System erhält. Der Zugriff kann auf bestimmte Tageszeiten beschränkt werden oder auch auf spezielle IP-Adressen. Gibt es keine Beschränkung bzgl. der IP-Adressen, wird grundsätzlich überprüft, ob von dieser IP-Adresse schon einmal eine Verbindung mit dem Salesforce-System existiert hat. Falls dem so ist, wird die Verbindung gewährt, ansonsten wird überprüft, ob die Anfrage von einem Browser mit gültigem Cookie kommt. Trifft dies ebenfalls nicht zu, wird überprüft, ob die IP-Adresse auf der Liste von vertrauenswürdigen IP-Adressen steht. Falls dies auch nicht der Fall ist, wird der Zugriff verweigert.

⇒ AZS-Bewertung: (+).

Die Benutzerauthentifizierung funktioniert immer gleich. Der Benutzer schickt seinen Benutzernamen und sein Passwort zum Service `login` und bekommt ein *Security Token* sowie eine *Endpoint URL* zurück. Diese Daten sind nun für alle folgenden API-Aufrufe zu benutzen. Dieser Vorgang ist sehr einfach, wenn der Client das Salesforce-eigene Web-Service-

Framework WSC einsetzt. In dem Fall wird der gesamte Authentifizierungscode automatisch generiert und der Entwickler muss sich keine Gedanken darüber machen, wie die Authentifizierung funktioniert. Fall eine andere Technologie (z.B. JAX-WS, BPEL, SOAP-UI) zum Einsatz kommt um die Services aufzurufen, muss der Entwickler den Algorithmus selbst implementieren. Dieser ist nicht sehr kompliziert, aber der Entwickler muss wissen, wie mit verschiedenen Technologien umzugehen ist (HTTP, SAAJ, JAXB, JAX-WS, DOM, XML).

⇒ EAM-Bewertung: (○).

3.3.1.4 Zusätzliche Informationen

Die Dokumentation ist auf der Webseite verfügbar, welche natürlich dazu gedacht ist, von Menschen gelesen zu werden. Eine zusätzliche maschinenlesbare Version wäre vorteilhaft.

Es gibt keine Dokumentation der Business-Objekte sowie deren Felder in den WSDL-Dokumenten selbst. Es wäre zu wünschen, wenn die Business-Objekte auch in der WSDL dokumentiert wären (in `<documentation>`-Tags) und wenn diese Dokumentation ebenfalls in die generierten Java-Stubs als JavaDoc übernommen würde.

⇒ DVA-Bewertung: (−).

Für Wertevorschläge gibt es drei verschiedene Datentypen. Zunächst gibt es die `<picklist>`, was einfach eine Menge von Alternativen enthält. Eine `<multipicklist>` ist fast das gleiche, nur dass mehrere Werte selektiert werden können. Der dritte Datentyp nennt sich `<combobox>`, was einer `<picklist>` entspricht, bei dem auch ein beliebiger Freitext eingegeben werden kann.

⇒ VAA-Bewertung: (+).

Um zu markieren, dass ein Parameter optional ist, wird das Attribut `minOccurs` in der WSDL eingesetzt. Hier sind mit nur wenigen Ausnahmen fast alle Daten optional (d.h. `minOccurs="0"`).

Die Taktik fast alle Datenfelder bis auf einige Schlüsselfelder als optional zu kennzeichnen ist eine sehr einfache Lösung, jedoch scheint sie dem Anwender nicht besonders gut zu helfen. Hier wird der Benutzer nicht besonders stark geführt.

⇒ DOP-Bewertung: (○).

Die API kann befragt werden nach Beschriftungen für die GUI, die in die jeweils richtige Sprache übersetzt sind. Somit ist die gesamte API internationalisiert. Zusätzlich gibt es sogar Layout-Informationen, d.h. Informationen darüber, wie die Daten in einer externen Applikation am

Bildschirm angeordnet werden sollte. Diese Funktion kann sehr hilfreich sein, besonders wenn man automatisch Benutzeroberflächen generieren möchte. Dann hat man z.B. die Information, dass die Hausnummer in der Adresse zur Straße gehört und im Optimalfall direkt dahinter angezeigt wird.

⇒ ISN-Bewertung: (+).

3.3.2 NetSuite

NetSuite [net12b] ist ein Cloud-basiertes vollständiges ERP-System, welches als SaaS-Lösung angeboten wird. Die Zielgruppe sind Unternehmen mittlerer Größe oder Teile großer Unternehmen. NetSuite bietet eine Web-Service-API mit Namen „SuiteTalk“ an. Diese API setzt auf den bekannten WSDL/SOAP-Technologie-Stack.

3.3.2.1 Zentrale Anforderungen

Wie die Salesforce-Lösung bietet NetSuite eine WSDL für die ganze Applikation an. Die Typ-Informationen (also die Informationen, wie die Business-Objekte strukturiert sind), sind in 42 externe XSD-Dokumente ausgelagert, die allesamt per `<import>`-Statement in die WSDL eingebunden sind. Nur eine WSDL zu benutzen macht eine extra Registry überflüssig, da die WSDL die Registry sozusagen selbst enthält. Möchte ein Client die Liste aller Business-Objekte auslesen (die hier „record types“ heißen), kann er entweder alle Elemente aus den XSD-Dateien heraussuchen, die den Typen `<record>` erweitern oder er benutzt die Enumeration „RecordType“, die eine Liste aller 131 Business-Objekte enthält. Die WSDL enthält insgesamt lediglich 23 Operationen. Alle Operationen, die sich auf Daten beziehen (wie z.B. add, get, update, delete, ...) können auf alle Business-Objekte angewandt werden. Diese müssen dazu entsprechend übergeben werden.

⇒ RSD-Bewertung: (+).

WSDLs und die automatische Generierung von Java-Code werden eingesetzt. Somit die die API-Definition Teil der Implementierung und ist immer korrekt und vollständig.

⇒ KVS-Bewertung: (+).

Alle Business-Objekte in NetSuite sind per API erreichbar. Es gibt nichts, was in der GUI möglich ist, aber nicht per API. Auf jedes Objekt kann dabei über jede CRUD-Operation zugegriffen werden. Die Abdeckung der Business-Objekte ist demnach sehr gut (45 von 50 Punkte), wie es in Tabelle 3.5 zu sehen ist.

Tabelle 3.5: NetSuite - Business-Objekte

	Create[a]	Read		Update	Delete
		Simple[b]	Query[c]		
Customer	✓	✓	✓	✓	✓
Vendor	✓	✓	✓	✓	✓
Item[d]	✓	✓	✓	✓	✓
Sales Quote[e]	✓	✓	✓	✓	✓
Sales Order	✓	✓	✓	✓	✓
Sales Invoice	✓	✓	✓	✓	✓
Purchase Quote	–	–	–	–	–
Purchase Order	✓	✓	✓	✓	✓
Purchase Invoice[f]	✓	✓	✓	✓	✓
Employee	✓	✓	✓	✓	✓

[a] hier: Add
[b] hier: Get
[c] hier: GetList
[d] hier: InventoryItem
[e] hier: Estimate
[f] hier: VendorBill

Das System bietet kein Business-Objekt für Angebote (also „Sales Quote" oder „Purchase Quote") an, jedoch gibt es das Business-Objekt „Estimate" (zu deutsch „Abschätzung" oder „Kalkulation") und das „NetSuite Help Center" definiert „Estimate" folgendermaßen:

> An estimate transaction, sometimes referred to as a quote, is a non-posted record of estimated charges to a customer.

Das Business-Objekt „Estimate" wird hier also synonym zu „Sales Quote" verwendet.

⇒ VZB-Bewertung: (+).

Die gesamte API besitzt eine einzige Versionsnummer. Die einzelnen Services sind nicht versioniert. Rückwärtskompatibilität wird versucht so gut wie möglich zu gewährleisten. Hinzufügungen wie neue Operationen oder „record types" (also Business-Objekte) bewirkten eine neue „minor revision". Der Namensraum wird dabei nicht verändert. Neue minor-Versionen unterstützen immer alle Funktionen der Vorgängerversion. Wenn Änderungen vorgenommen werden, die dazu führen, dass Rückwärtskompatibilität nicht mehr gewährleistet werden kann (z.B. durch Hinzufügen eines neuen Pflichtfeldes) wird die Versionsnummer im Namensraum geändert und alle Applikationen, die die API benutzen müssen entsprechend angepasst werden. Es gibt keine Garantie, dass ein neues „major release" alle vorherigen Versionen unterstützt.

⇒ SDD-Bewertung: (○).

3.3.2.2 API-Design

Die Benennungen in der API sind zwar im Allgemeinen relativ intuitiv gewählt, aber leider nicht immer konsistent. Ein Beispiel für eine Inkonsistenz ist die Benennung eines Artikels (engl. „Item"). In der API wird dieses „InventoryItem" genannt, wohingegen es in der GUI einfach „Item" heißt. Die weiter oben schon erwähnten „estimates" sind ein weiteres Beispiel. Eigentlich sind „estimate" und „quote" nicht genau synonym, NetSuite behandelt sie jedoch gleich. Auf der NetSuite-Homepage schreiben sie über „Quote and Order Management", aber die API kennt nur „estimate". Nicht nur Benennungen sind inkonsistent. Für die meisten Objekte gibt es eine „Entity-ID", also einen eindeutigen Identifizierungsschlüssel für dieses Objekt. Leider besitzen nicht alle Business-Objekte dieses Feld, so hat ein „InventoryItem" zum Beispiel eine „Item-ID".

⇒ IKB-Bewertung: (○).

Die Struktur der API ist klar und einfach gehalten.

⇒ KES-Bewertung: (+).

Die API-Technologie ist sehr gut dokumentiert. Die Business-Objekte und deren Felder sind ebenfalls gut dokumentiert, und zwar in einem PDF-Dokument, welches auf der NetSuite-Webseite zum Herunterladen bereitsteht [net12a].

⇒ KVD-Bewertung: (+).

3.3.2.3 Technologie

Mit WSDL und SOAP werden plattformunabhängige Standards eingesetzt, so dass die API von beliebigen Plattformen und Programmiersprachen aus benutzt werden kann.

⇒ PUS-Bewertung: (+).

Der Datentransfer ist grundsätzlich verschlüsselt. Dies wird durch den strikten Einsatz von HTTPS (SSL) mit 128-Bit-Verschlüsselung erreicht. Alle Web-Services verlangen Authentifizierung und Autorisierung. Ein Client bekommt Zugang zur API durch den Aufruf des Login-Services. Dieser Service erzeugt eine Session, welche automatisch nach 15 Minuten ohne Aktivität geschlossen wird. Es existiert grundsätzlich ein Limit von zwei Verbindungen pro Benutzername-Passwort-Kombination: eine Verbindung für die GUI und eine zweite für die API.

⇒ AZS-Bewertung: (+).

Die Benutzerauthentifizierung wird durch den Login-Service durchgeführt. Der Benutzer sendet dazu seinen Benutzernamen, sein Passwort, seine „Account-ID“ und optional noch seine Rolle zum Server. Danach erzeugt der Service eine Session und gibt die dazugehörige Session-ID als Cookie zurück an den Client. Dieser Cookie wird nun für alle folgenden Service-Aufrufe benutzt. Der Logout-Service schließt die Session.

⇒ EAM-Bewertung: (◦).

3.3.2.4 Zusätzliche Informationen

Die Dokumentation ist als PDF-Dokument verfügbar, welches von der NetSuite-Seite heruntergeladen werden kann. Dieses ist natürlich dazu gedacht, von Menschen gelesen zu werden. Eine zusätzliche Maschinenlesbare Version wäre von Vorteil. Es gibt keine Dokumentation der Business-Objekte oder deren Felder in den WSDL-Dokumenten selbst.

⇒ DVA-Bewertung: (−).

Für Wertevorschläge setzt NetSuite das XSD-Element `<enumeration>` ein.

⇒ VVA-Bewertung: (+).

In den XML-Schema-Dokumenten wird das Attribut `minOccurs` zur Kennzeichnung optionaler Elemente eingesetzt. Dies ist jedoch fast immer auf „0“ gesetzt, so dass alle Parameter als optional gelten. Die einzigen XSD-Dokumente, die Pflichtelemente enthalten (`minOccurs="1"`) sind drei (eher technische) Kern-Dokumente[20] (von 43 Dokumenten insgesamt). Die Business-Objekte haben allesamt keinerlei verpflichtende Bestandteile.

⇒ DOP-Bewertung: (−).

3.3.3 Workday

Workday [wor12] ist eine SaaS-Lösung, welche nicht wirklich alle Bereiche von dem abdeckt, was man unter einem ERP-System versteht. Stattdessen konzentriert sich die Software auf die Mitarbeiter eines Unternehmens, weswegen sie sich selbst auch „Human Capital Management Tool“ nennt. Neben der Verwaltung von Mitarbeitern werden ebenfalls thematisch verwandte Gebiete abgedeckt wie z.B. Gehaltsabrechnungen, Ausgabenverwaltung, Projektmanagement und Finanzmanagement. Workday wurde 2005 von Dave Duffield (Gründer von PeopleSoft) und Aneel Bhusi (ehemaliger Vize-Präsident von PeopleSoft) gegründet.

3.3.3.1 Zentrale Anforderungen

Das Workday-System besitzt keine echte Registry, welche aber auch nicht wirklich benötigt wird. Es gibt nur eine feste Anzahl von WSDL-Dokumenten, genau eine WSDL pro Kategorie von Services. Es ist ebenfalls möglich, mehrere dieser WSDLs zu größeren zu bündeln durch das Hintereinanderhängen von durch Plus-Zeichen getrennten Kategorienamen in der URL der WSDL. Wenn also der Client auf die gesamte API zugreifen möchte, so ist es theoretisch auch möglich, eine WSDL für das gesamte System herunterzuladen und anschließend z.B. Java-Stubs daraus zu generieren.

Innerhalb der Kategorien existieren aus einer rein technischen Sicht keine weiteren Gruppierungen oder Kategorisierungen der Services mehr. Auch das Konzept von Business-Objekten gibt es aus technischer Sicht hier nicht. Die richtige Funktion kann hier dadurch gefunden werden, dass sich die Service-Namen an ein konsistentes Namensschema halten. Alle Service-Namen beginnen mit einem Präfix, wobei es sich immer um ein Verb handelt, welches angibt, was mit einem Business-Objekt getan wird (z.B. „Get“ zum Lesen von Daten oder „Add“ zum Hinzufügen

[20] `activities.scheduling.xsd`, `platform.core.xsd` und `plaform.massages.xsd`

von Daten). Nach einem Unterstrich als Trennsymbol folgt der Name des Business-Objektes. Die WSDL-Dateien sind recht groß, da es eine spezifische Funktion für jeden Service gibt (z.B. „Find_Employee“ oder „Get_Job_Profiles“).

Die Kategorien gruppieren nicht wirklich die Business-Objekte, sondern die Funktionen. Das bedeutet, dass ein Business-Objekt mit mehreren seiner Methoden in der einen Kategorie und mit einigen anderen Methoden in einer zweiten Kategorie vertreten sein kann. Dies macht es manchmal schwieriger, den richtigen Service zu finden. Zum Beispiel sind die Services „Hire_Employee“ und „Terminate_Employee“ in der Kategorie „Staffing“ einsortiert, „Get_Employee“ und „Find_Employee“ jedoch unter „Human Ressources“. Methoden von „Applicant“ können teilweise in „Recruiting“ und teilweise in „Staffing“ gefunden werden.

⇒ RSD-Bewertung: (○).

Die Schnittstellen werden durch WSDL-Dateien beschrieben. Dies ist zwar eine sehr technische Sicht auf die Services, aber sie ist immer korrekt, da die aktuelle Implementierung direkt auf dieser Information basiert.

⇒ KVS-Bewertung: (+).

Die allgemeine Situation bzgl. der Abdeckung von Business-Objekten ist sehr vergleichbar zu Salesforce. Alle Business-Objekte der Workday-Software können per API erreicht werden. Es gibt kein Objekt in der GUI, welches nicht in einer WSDL auftaucht. Dies ist in Anbetracht des limitierten Anwendungsgebietes der Software vergleichsweise einfach zu erreichen. Auf jedes Objekt kann durch eine bestimmte Menge an Operationen zugegriffen werden. Anders als bei Salesforce, kann hier nicht einfach jede Operation mit jedem beliebigen Business-Objekt kombiniert werden. Stattdessen besitzt jedes Business-Objekt seine ganz spezifische Menge an Operationen.

Leider gibt es keine Funktion (also kein spezieller Präfix) für komplexe Abfragen. Manchmal stellen Funktionen mit „get“ einfache Leseoperationen dar und manchmal weisen sie Eigenschaften komplexer Abfragen auf. So ist es im Allgemeinen nicht einfach möglich, für eine bestimmte Abfrage eine Liste von Objekten zu bekommen. Wenn der Client nach einem bestimmten Objekt sucht, kann eine Funktion mit dem Präfix „find“ benutzt werden.

Da Workday kein vollständiges ERP-System, sondern „nur“ ein HCM-System ist, wäre es nicht richtig und fair die gleiche Menge an Business-Objekten wie z.B. bei SAP für die Untersuchungen zu benutzen. Die Untersuchung auf Basis einer alternativen Menge von Business-Objekten ist in Tabelle 3.6 zu sehen. Bei 23 von 35 erreichten Punkten ist diese

Abdeckung noch positiv zu bewerten[21].

⇒ VZB-Bewertung: (+).

Wie bei Salesforce besitzt die gesamte API eine Versionsnummer. Obwohl die Workday-Lösung immer noch sehr jung ist (2005) - im Vergleich zu z.B. SAP-ERP - steht die aktuelle Versionsnummer bereits bei „v15“. Jede WSDL (also jede Kategorie) wird separat versioniert, es ist also möglich eine Versionsnummer für die eine Kategorie und eine andere für eine andere Kategorie zu nehmen. Bei jedem Request muss die API-Versionsnummer mitgesendet werden[22]. Workday nennt dies das „Version Parameter Design Pattern“. Es ist ebenfalls möglich, die Versionsnummer in den URL des Services mit aufzunehmen. Dies wird „Version Endpoint Design Pattern“ genannt. Die Wahl, welches Design Pattern man nehmen sollte ist recht technischer Natur. In beiden Fällen nennt der Client dem Server, welche Version zu benutzen ist. Neue Versionen werden eingeführt, wenn Änderungen nicht rückwärtskompatibel sind. Rückwärtskompatible Änderungen sind z.B. das Hinzufügen neuer Operationen oder neue Datentypen, rückwärtsinkompatibel sind z.B. das Entfernen oder das Umbenennen von Operationen, das Ändern von Parametern einer Operation oder das Ändern der Struktur eines komplexen Datentyps. Operationen werden normalerweise nicht direkt gelöscht, sondern stattdessen als „deprecated“ markiert. Nach einer Weile werden sie dann schließlich irgendwann gelöscht.

⇒ SSD-Bewertung: (◦).

3.3.3.2 API-Design

Leider ist das Benennungsschema der Workday-API nicht sehr konsistent. Zum Beispiel gibt es zum einen Operationen mit „Add_Update“ als Präfix und zum anderen „Put“-Operationen. Beide führen exakt dasselbe durch: Sie überprüfen zunächst, ob ein Objekt existiert. Wenn ja, wird es aktualisiert, wenn nicht, neu angelegt. Ein anderes Beispiel ist, dass es manchmal für sehr ähnliche Aktionen verschieden benannte Operationen für verschiedene Business-Objekte gibt. So gibt es zum Beispiel sehr viele verschiedene Präfixe, die alle irgendwie mit dem Löschen zu tun haben: „Cancel“, „Dissolve“, „End“, „Inactivate“, „Remove“, „Rescind“, „Terminate“, „Unpost“. Hier könnte man natürlich immer noch argumentieren, dass es hier (wenn auch manchmal subtile) Unterschiede in den Bedeutungen gibt. Dies wird jedoch schon schwieriger, wenn man auf die Services zum Ändern des Datenbestands schaut: „Adjust“,

[21] Angepasstes Bewertungsschema: 28-35: (+), 18-27: (◦), 0-17: (−)
[22] https://community.workday.com/node/204

Tabelle 3.6: Workday - Business-Objekte

	Create	Read		Update	Delete
		Simple	Query		
Employee	✓[a]	✓[b]	–	✓[c]	✓[d]
Applicant	✓[e]	✓/–[f]	✓[g]	✓[h]	–
Bank Account	✓[i]	–	✓[j]	✓[k]	–
Business Unit	✓[l]	–	✓[m]	✓[n]	–
Basic Project	✓[o]	–	✓[p]	✓[q]	–
Job Profile	✓[r]	✓/–[s]	✓[t]	✓[u]	–
Position	✓[v]	–	✓[w]	✓[x]	–

[a]hier: Hire_Employee
[b]hier: Get_Employee
[c]hier: Update_Employee_Personal_Info, Update_Employee_Image, Demote_Employee,...
[d]hier: Terminate_Employee
[e]hier:Put_Applicant
[f]hier: Get_Applicant, jedoch „deprecated“, Get_Applicants stattdessen empfohlen
[g]hier: Get_Applicants
[h]hier: Put_Applicant
[i]hier: Put_Bank_Account
[j]Get_Bank_Accounts
[k]Put_Bank_Account
[l]hier: Put_Business_Unit
[m]Get_Business_Units
[n]Put_Business_Unit
[o]hier: Put_Basic_Project
[p]Get_Basic_Projects
[q]Put_Basic_Project
[r]hier:Put_Job_Profile
[s]hier: Get_Job_Profile, jedoch „deprecated“, Get_Job_Profiles stattdessen empfohlen
[t]hier: Get_Job_Profiles
[u]hier: Put_Job_Profile
[v]hier: Put_Position
[w]Get_Positions
[x]Put_Position

„Change“, „Edit“, „Maintain“, „Manage“, „Update“. Aufgrund der Tatsache, das diese Flut von Benennungen für ähnliche Aktionen der einzige negative Punkt im Benennungsschema ist und für jede Operation relativ klar ist, was dahinter steckt, kann hier noch eine mittlere Bewertung gerechtfertigt werden.

⇒ IKB-Bewertung: (○).

Um es einfacher zu machen, den richtigen Service zu finden, sind sie in Kategorien einsortiert, die jeweils ein WSDL-Dokument besitzen. Relativ ungewöhnlich ist die Tatsache, dass manche Business-Objekte auf verschiedene Kategorien aufgeteilt sind. Manchmal ist es sogar so, dass ein bestimmter Service gleich in mehreren Kategorien auftaucht.

⇒ KES-Bewertung: (○).

Die Dokumentation ist sehr gut und scheint ebenfalls komplett zu sein. Fehler konnten nicht gefunden werden.

⇒ KVD-Bewertung: (+).

3.3.3.3 Technologie

Workday setzt die plattformunabhängigen Standards WSDL und SOAP ein.

⇒ PUS-Bewertung: (+).

Die Sicherheit der Services wird gewährleistet durch den strikten Einsatz von HTTPS (SSLv3, TLS). Außerdem wird ein System zur „Perimeter-level defense and network intrusion prevention“ eingesetzt. Sogar die Werte in der Datenbank werden verschlüsselt (AES, 256 Bit). Workday hat ein Benutzerrechtemodell, welches sie selbst als konfigurierbar und prüffähig („auditable“) bezeichnen. Es ist zertifiziert durch Audits wie ISO 27001, SAS70 Type II und Safe Harbor[23] .

⇒ AZS-Bewertung: (+).

Leider hatten wir für diese Evaluierung kein laufendes Workday-System zur Verfügung, sondern konnten uns nur auf die Informationen der Dokumentation und die Workday-Webseite berufen. Daher konnte auch die Authentifizierungsmethode nicht überprüft werden.

3.3.3.4 Zusätzliche Informationen

Die Workday-WSDL-Dokumente machen sehr häufig Gebrauch des Tags `<documentation>`, wodurch Dokumentation zu allen Operationen und

[23] Alle Fakten dieses Abschnittes können unter http://www.workday.com/why_workday/technology/security.php nachgelesen werden.

all ihren Parametern angeboten wird.

⇒ DVA-Bewertung: (+).

Für Wertevorschläge benutzt Workday ein spezielles <enumeration>-Element aus dem Workday-XML-Namensraum.

⇒ WVA-Bewertung: (+).

Es werden Kardinalitäten für alle Parameter angegeben (minOccurs und maxOccurs in der WSDL), aber diese ist immer [0..1] oder [0..*], so dass alle Parameter optional sind. Sogar in den get-Operationen ist der Referenz-Parameter (welcher die ID enthält) als optional gekennzeichnet. In der gesamten API gibt es nicht einen Eintrag mit minOccurs="1".

⇒ DOP-Bewertung: (−).

Es gibt Validierungsregeln sowohl in der Dokumentation auf der Workday-Homepage als auch in der WSDL. Dafür existiert ein spezieller Tag <validation>[24]. Die Validierungsregeln sind in normalem Englisch, nicht in einer formalen Sprache notiert.

⇒ VRA-Bewertung: (+).

3.4 Vergleich

In diesem Abschnitt werden die oben beschriebenen Ergebnisse zusammengefasst und nebeneinandergestellt. Zunächst wird in Abschnitt 3.4.1 erläutert, welche allgemeinen Konzepte verfolgt wurden und somit wie sich die verschiedenen Lösungen grob klassifizieren lassen. Anschließend werden in Abschnitt 3.4.2 die untersuchten APIs jeweils ganz konkret bewertet und nebeneinandergestellt.

3.4.1 Allgemeine Konzepte

Eines haben alle Lösungen gemeinsam: Es wird grundsätzlich über etwas wie „Business-Objekte“ gesprochen. Diese Objekte entsprechen ungefähr den Tabellen einer relationalen Datenbank oder - um es genauer zu sagen - den Entities in einem Entity-Relationship-Modell [Che76]. Die Einführung des Konzeptes der Business-Objekte um Funktionen bzw. Services zu gruppieren ist sehr natürlich und kann auch von Menschen ohne IT-Fachwissen verstanden werden. Außerdem muss erwähnt werden, dass diese Gruppierung keine Einschränkung der Mächtigkeit bedeutet. Jegliche Daten oder Funktionen können durch eine Operation bereitgestellt werden, die zu einem Business-Objekt gehört, selbst komplexe Abfragen,

[24] im proprietären Workday-Namensraum: xmlns:wd:="urn:com.workday/bsvc"

die auch Beziehungen zwischen mehreren Datenbank-Entities berücksichtigen. In diesem Fall muss der Designer der API entscheiden, zu welchem Business-Objekt diese Funktion am besten passt. Alternativ kann auch für solche Zwecke ein zusätzliches „künstliches" Business-Objekt eingeführt werden, welches also nicht direkt nur einer Datenbank-Tabelle entspricht.

Obwohl es immer „Business-Objekte" in den APIs der untersuchten Produkte gibt, kann die Art und Weise sehr unterschiedlich sein, wie diese organisiert und technisch umgesetzt werden. Im allgemeinen können hier zwei grundsätzliche Herangehensweisen unterschieden werden:

1. **Funktionen** als Hauptordnungselemente
2. **Business-Objekte** als Hauptordnungselemente

SAP und Workday benutzen zum Beispiel die erste Herangehensweise. Hier sind Business-Objekte lediglich in der Dokumentation existent. Außerdem können die Benennungen der Business-Objekte noch Auswirkungen auf das Benennungsschema der Funktionen haben. Die API selbst ist aus rein technischer Sicht eine große aber flach strukturierte Sammlung von Funktionen.

Dynamics NAV und NetSuite wählen den zweiten Ansatz. Diesen könnte man auch als „algebraisch" bezeichnen, da es auf der einen Seite Mengen von Elementen gibt (hier also die Business-Objekte wie Kunden oder Mitarbeiter) und auf der anderen Seite eine Menge von Operationen, die auf die Business-Objekte angewandt werden können (z.B. „Create", „Read", „Update", „Delete", ...).

Beide Ansätze haben ihre Vor- und Nachteile. Der Funktionen-Ansatz macht es möglich individuelle Namen für alle Funktionen zu vergeben. Somit kann die Benennung nah am Business-Vokabular sein. Zum Beispiel kann ein Mitarbeiter eigentlich nur eingestellt („hire") und nicht erzeugt („create") werden, `hire_employee` ist demnach eine treffendere Bezeichnung als `create_employee`. Weiterhin ist die Versionierung der API einfacher, da jede einzelne Funktion für sich einzeln versioniert werden kann. Der Business-Objekt-Ansatz führt auf der anderen Seite zu einer konsistenteren und besser strukturierten API. Ein Benutzer muss hier nur eine relativ kleine Menge an Operationen kennen und schon versteht er dadurch sehr gut, was genau ein bestimmter Service an Funktionalität anbietet. Außerdem ist es hier einfacher eine gute Abdeckung der benötigten Funktionen zu gewährleisten, da man relativ leicht erkennen kann, welche Operationen für ein bestimmtes Business-Objekt angeboten werden. Es ist bei diesem Ansatz insgesamt einfacher, einen guten Überblick über die API zu behalten.

Aufgrund der Tatsache, dass SOAP-Web-Services die Standardtechnologie zur Realisierung von Business-APIs darstellen war es natürlich, dass in dieser Untersuchung der Großteil der Lösungen auf diese Technologie aufsetzte. Obwohl hier jeweils die gleiche Technologie eingesetzt wurde, kann man erkennen, dass die konkrete Umsetzung sehr unterschiedlich sein kann. Zusätzlich zu der Tatsache, dass die Lösungen verschiedene Ansätze zur Strukturierung einsetzen (siehe oben: Funktionen oder Business-Objekte als Hauptordnungselemente), wurden auch die WSDL-Dokumente sehr unterschiedlich gestaltet. Die Lösung von Salesforce liefert eine riesige WSDL-Datei für die gesamte API, Workday besitzt ein WSDL-Dokument pro Kategorie (ein Porttype pro Kategorie, mit der Möglichkeit, die Dokumente zu größeren zu kombinieren) und schließlich liefert SAP eine WSDL pro Funktion aus. Ein WSDL-Dokument besitzt hier grundsätzlich nur einen Porttype mit einer einzigen Funktion. Es ist sogar möglich, sich pro Funktion drei verschiedene WSDL-Dokumente ausliefern zu lassen, die dann jeweils nur einen Aspekt einer WSDL bzw. der Funktion beinhalten (1. Interface, 2. Binding, 3. Service-Endpoint).

3.4.2 Vergleich der APIs

Wie man in den Tabellen 3.7 und 3.8 leicht sehen kann, sind alle untersuchten Lösungen relativ weit davon entfernt, alle Anforderungen zu erfüllen. Die Ergebnisse spiegeln sehr gut wieder, wie die hinter den Produkten stehenden Unternehmen jeweils ihr Geschäftsprofil ausrichten.

SAP hat mit der Einführung der eSOA-Technologie gegenüber dem alten BAPI-Standard einen deutlichen Schritt nach vorne gemacht. Das proprietäre RFC-Protokoll wurde durch plattformunabhängige Web-Services ersetzt und die Anzahl der angebotenen Services wuchs gewaltig. Nichtsdestotrotz ist die Abdeckung der Business-Objekt-Operationen bei beiden SAP-Lösungen immer noch so schlecht, dass Kunden die Service-Sammlungen selbst erweitern müssen, meist durch die Hilfe von SAP-Beratern. Folglich kommt es immer noch zu den SAP-typischen, oft langdauernden und teuren Customizing-Phasen, was zeigt, dass sich das SAP-Business-Modell, welches hauptsächlich auf mittlere bis große Unternehmen als Kunden abzielt, kaum geändert hat. Dennoch erleichtert eSOA das Customizing und kann als ein Schritt auch in Richtung der mittelgroßen Unternehmen gesehen werden. Die Stärke beider SAP-Lösungen liegt in der Stabilität der APIs. Sowohl BAPI als auch eSOA besitzen sehr stabile Definitionen ihrer Services. Ein Client, der eine dieser APIs einsetzt, kann sich ziemlich sicher sein, dass diese Lösung auch noch in ferner Zukunft genau so funktioniert.

Die API von Dynamics NAV hat einen grundsätzlich anderen Charakter. Was zunächst ins Auge fällt ist die besonders gute Abdeckung von einfa-

Tabelle 3.7: Ergebnisse der Auswertung - traditionelle „on-premise“ ERP-Systeme

	BAPI	eSOA	NAV	QBDE
RSD	−	+	○	○
KVS	○	+	+	○
VZB	−	○	+	−
EDV	+	+	+	+
SSD	+	+	−	−
IKB	○	○	+	+
KES	○	−	+	+
KVD	−	○	−	○
PUS	○	+	○	−
AZS	○	○	○	○
EAM	+	+	+	○
GLD	*n.u.*[a]	*n.u.*	*n.u.*	*n.u.*
DVA	+	−	−	○
WVA	+	−	+	+
DOP	−	+	+	+
VVA	−	−	−	−
GIS	−	−	−	−
ISN	−	−	−	−

[a] nicht untersucht

chen CRUD-Operationen für alle Business-Objekte. Dies kann für viele kleine und mittlere Unternehmen schon ausreichen und kann zu einer sehr schnellen und kostengünstigen Einführung dieser Software führen. Außerdem ist die API sehr gut strukturiert, was es recht einfach macht, mit ihr zu arbeiten. Leider wird es in dem Moment schwierig, in dem man die Service-Sammlung selbst erweitern möchte, da die NAV-Lösung kaum etwas zum Management von neuen Services anbietet. Ein weiterer Punkt ist, dass die Funktionen nicht wirklich als API veröffentlicht

Tabelle 3.8: Ergebnisse der Auswertung - Cloud-basierte ERP-Systeme

	Salesforce	NetSuite	Workday	QBOE
RSD	+	+	○	○
KVS	+	+	+	○
VZB	+	+	○	−
EDV	+	+	*n.u.*[a]	+
SSD	○	○	○	−
IKB	+	○	○	+
KES	+	+	○	+
KVD	+	+	+	○
PUS	+	+	+	○
AZS	+	+	+	+
EAM	○	○	*n.e.*	−
GLD	*n.u.*	*n.u.*	*n.u.*	*n.u.*
DVA	−	−	+	○
WVA	+	+	+	+
DOP	+	−	−	+
VVA	−	−	+	−
GIS	−	−	−	−
ISN	+	−	−	−

[a] nicht untersucht

werden. Es existiert zum Beispiel keinerlei Versionierung - weder von einzelnen Services noch von der API im Ganzen. Somit weiß der Benutzer nie, ob eine Client-Lösung noch mit dem nächsten Release von Dynamics NAV zusammenarbeitet. Der Ansatz von Dynamics NAV kann also für kleine bis mittlere Unternehmen empfohlen werden, die lediglich Standard-Anforderungen an die Software stellen.

Die Quickbooks-API punktet mit ihrer relativ guten „Online Reference“. Hier werden alle Services in verschiedenen Formaten beschrieben und mit

relativ ausführlicher Dokumentation präsentiert. Solang die Zahl der Services nicht zu groß wird, ist diese Lösung recht praktisch. Bei einer größeren Menge an Services wird das ganze schnell unübersichtlich, da wie bei Dynamics NAV jegliche Form von weiterer Kategorisierung und Strukturierung fehlt. Die zugrundeliegende Technologie der Quickbooks-API ist sehr speziell und kann nur in reinen Windows-Umgebungen auf eine einfache Weise benutzt werden. Somit wird deutlich, warum Quickbooks als eine „Low-Budget"-Lösung angesehen werden kann, die nur für sehr kleine Unternehmen mit Standard-Anforderungen ausreichend ist.

Die APIs von Salesforce und NetSuite haben sehr viel gemeinsam. Beide sind insgesamt qualitativ sehr hochwertig, besonders was die zentralen Anforderungen an Business-APIs und das API-Design angeht. Salesforce hat dazu noch als positives Alleinstellungsmerkmal die Bereitstellung internationalisierter Labels sowie Layout-Informationen für graphische Benutzeroberflächen, was es recht einfach macht, brauchbare eigene Oberflächen zu generieren. Salesforce und NetSuite sind beides recht moderne Systeme (z.B. im Vergleich zu SAP oder Dynamics NAV). Da sie nicht mit Legacy-Strukturen und Rückwärtskompatibilität zu kämpfen haben, konnten die Entwickler dieser Systeme von Grund auf moderne APIs entwerfen. Außerdem haben sie erkannt, dass eine API heutzutage mindestens genauso wichtig ist wie die graphische Benutzeroberfläche. Wenn man nur die jeweiligen APIs betrachtet, können diese beiden Systeme relativ uneingeschränkt empfohlen werden.

Die API von Workday besitzt eine besonders gute Dokumentation, die sogar auch komplett über die API selbst (also über die WSDL-Dokumente) zugänglich ist. Hiermit steht diese API allein da. Außerdem ist es die einzige API, die auch Validierungsregeln für die einzelnen Services mit ausliefert. Leider hatten wir keine laufende Test-Instanz von Workday zur Verfügung, so dass wir selbst keine Experimente durchführen konnten. So ist es uns nicht möglich, ein abschließendes Urteil über die Software zu fällen.

3.5 Schnittstellen weiterer Produkte

Neben den oben beschriebenen Business-APIs, gibt es noch weitere APIs, die im Business-Kontext eine Rolle spielen, interessante Eigenschaften und Eigenheiten aufweisen, und somit Wert sind in diesem Zusammenhang betrachtet zu werden.

Zum einen gibt es immer mehr Open-Source-Implementierungen von ERP-Systemen, deren APIs in Abschnitt 3.2 behandelt werden. Da es sich um ERP-APIs handelt, fallen diese APIs auch in die Kategorie der

Business-APIs. In Abschnitt 3.5.2 folgt dann noch eine Betrachtung der APIs der E-Commerce-Plattformen Amazon und EBay, welche durch ihre Domäne und ihre Komplexität ebenfalls als Business-APIs bezeichnet werden können.

Weitere Systeme und deren APIs werden im Kapitel 5 behandelt, wo es um die Service-Integration im jABC geht. Dort wird unter anderem gezeigt, wie Office-Suiten oder die Web-Applikationen von Google eingebunden wurden und wo dort die jeweiligen Besonderheiten liegen.

3.5.1 OpenSource-ERP-Systeme

Open-Source-Lösungen führen im ERP-Bereich eher ein Nischendasein. Aber gerade für kleine und mittlere Unternehmen mit begrenzter Kapitalmacht bieten diese Systeme eine große Chance. Die Akzeptanz in Unternehmen von Open-Source im Allgemeinen und Open-Source-ERPs im Speziellen ist schon seit langem steigend [Car06]. Mittlerweile existiert eine ganze Reihe unterschiedlicher Lösungen, so dass eine gewisse Auswahl vorhanden ist. Diese Lösungen unterscheiden sich untereinander erheblich und auch bezüglich des Themas Service-Integration werden sehr unterschiedliche Konzepte verfolgt. In diesem Abschnitt wird zunächst kurz auf eine Reihe unterschiedlicher Systeme eingegangen und dann etwas ausführlicher auf die Systeme OpenERP und Apache Ofbiz. Die Untersuchungen zu den zuletzt genannten Systemen wurden mit der Unterstützung des Studenten Dominic Wirkner durchgeführt.

JFire [Nig13] ist eine auf Java-EE[25] basierende Lösung, die als Client auf die Eclipse-Plattform aufsetzt. Eine API für JFire wird über Java-RMI[26] und SOAP angeboten[27]. Leider ist zum Zeitpunkt der Fertigstellung dieser Arbeit keine Dokumentation der API online verfügbar, was gerade für ein Open-Source-Projekt besonders nachteilig ist und auf ernstzunehmende Probleme bei diesem Projekt hindeutet.

Compiere [Con13] ist zwar ebenfalls eine Open-Source-Lösung, sie bietet eine Web-Service-API jedoch nicht in der freien Version, sondern nur in einer „Enterprise Edition“ an, welche 995 Dollar pro Person und Jahr kostet. Hier wird das Thema Integration also als Kaufanreiz genutzt.

Der Compiere-Fork ADempiere [ADe13] besitzt ein Unterprojekt namens „ADempiere Web Services“, welches die öffentliche API des Projektes definiert und bereitstellt. Dazu ist folgendes zu lesen[28]:

[25]Konkret werden auf Serverseite die Technologien EJB 3 und JDO 3 eingesetzt.
[26]RMI: Remote Method Invocation
[27]https://www.jfire.org/modules/phpwiki/index.php/Architecture
2829

> „The idea is to integrate this project in future to ADempiere stable version.“

Demnach ist das Projekt momentan noch nicht in einer stabilen Version verfügbar.

Das ERP-Projekt xTuple [xTu13], welches sich in seiner freien Variante „Postbooks“ und in seiner kommerziellen Variante „OpenMFG“ nennt, wurde mit C++/Qt entwickelt und setzt als Datenbank ein PostgreSQL ein. Die API des Projektes[30], ist eine technische Besonderheit, da hier auf Datenbank-Views und SQL aufgesetzt wird. In den Dokumentationen ist entsprechend dokumentiert, über welche SQL-Statements man an die gewünschten Informationen gelangt oder entsprechende Datenmanipulationen vornimmt. Dieser Integrationsansatz ist dabei äußerst kritisch zu betrachten und auch nur mit Abstrichen überhaupt als „API“ zu bezeichnen. Für eine einfache Integration, wie sie in Abschnitt 5.3.5 beschrieben wird, ist diese API ungeeignet.

Die kleine PHP-basierte Lösung webERP bietet eine API über XML-RPC [Dav00] an[31], eine sehr einfach gehaltene, plattformunabhängige RPC-Lösung, die (wie SOAP) XML über HTTP versendet. XML-RPC ist quasi als Vorgänger von SOAP zu sehen, ist jedoch deutlich schlanker und einfacher gehalten. Die Einfachheit bedeutet jedoch auch, dass es hier kein Pendant zu WSDL gibt, es existiert also keine formale Schnittstellendefinition. Auch dies verhindert eine elegante und automatisierte Nutzung auf Client-Seite, gerade bei großen und komplexen APIs.

Open Bravo [Ope13a], ein weiterer Fork von Compiere, bietet gar keine öffentlichen Informationen zu einer API an. Gleiches gilt für das Projekt „Synerpy AvERP“ [Syn13].

OpenERP OpenERP [Ope13b] ist ein in Python implementiertes Open-Source-ERP-System. Die Daten werden dabei im Datenbanksystem PostgreSQL abgelegt. Als technische Grundlage für das ERP-System dient ein eigenes ORM-Framework[32] namens „OpenObject“. Das System besteht aus vielen verschiedene Untermodulen, die in einem öffentlichen Repository frei verfügbar sind. Auch die Entwicklung eigener Module ist möglich. Module können dabei eigene Business-Objekte definieren oder auch vorhandene Objekte erweitern. Business-Objekte werden in OpenERP durch Python-Klassen realisiert, die von einer bestimmten Klasse erben müssen[33]. Diese Klasse stellte alle grundlegenden Funktionen für

[30] http://www.xtuple.org/node/310 bzw. http://sourceforge.net/projects/postbooks/files/09%20PostBooks-API/
[31] http://www.weberp.org/wiki/APIDocumentation
[32] ORM: Object-relational mapping
[33] Klasse `osv`, kurz für „Object-Service“

Business-Objekte zur Verfügung, wie z. B. das OR-Mapping oder die typischen CRUD-Funktionen. Auf die Business-Objekte kann über XML-RPC [Dav00] (oder alternativ über das Python-spezifische NET-RPC) von außen zugegriffen werden. Hier wird OpenERP in Version 7 betrachtet.

Eine Registry ist bei OpenERP nicht vorhanden[34]. Für den korrekten Aufruf eines Services müssen der Name des Business-Objektes und die Bezeichnung der gewünschten CRUD-Operation bekannt sein. Manche Parameter sind immer gleich und fest vorgegeben. Die Namen der Datenfelder lassen sich über die Methode `fields_get` auslesen[35]. Grundsätzlich stehen für alle Business-Objekte alle CRUD-Operationen zur Verfügung[36]. Eine Validierung der Eingabedaten wird nur sehr begrenzt vorgenommen. Zum Beispiel werden Zeichenketten bei der Überschreitung der zulässigen Länge einfach abgeschnitten und Daten, die als „read-only“ gekennzeichnet sind, können zwar nicht verändert werden, ein Schreibversuch führt jedoch nicht zu einer Fehlermeldung[37]. Eine Versionierung der API ist nicht vorhanden[38].

Die Namensgebung ist intuitiv[39] und die Strukturen sind mit den immer gleichen CRUD-Operationen und übersichtlich aufgebauten Business-Objekten insgesamt klar, einfach und konsistent[40]. Die Dokumentation ist zwar recht umfangreich, leider ist sie jedoch stellenweise veraltet, beschreibt also nicht die aktuelle Version. Auch ist hier keine vollständige Liste der Business-Objekte enthalten[41].

Mit XML-RPC über HTTP wurde eine plattformunabhängige Technologie gewählt[42], eine Verschlüsselung ist durch die Nutzung von HTTPS möglich[43]. Dabei wird jeder Aufruf authentifiziert. Dazu ist die Angabe von Benutzername und Passwort nötig[44]. Zugriffsrechte werden dem User anhand seiner zugewiesenen Rollen zugeteilt[45].

Über spezielle Methoden werden nicht nur die Namen der Datenfelder verfügbar gemacht, sondern auch die dazugehörige Dokumentation[46]. Bei Eingabedaten mit Auswahlmöglichkeiten werden die Alternativen expli-

[34] RSD: (−)
[35] KVS: (+)
[36] VZB: (+)
[37] EDV: (○)
[38] SSD: (−)
[39] IKB: (+)
[40] KES: (+)
[41] KVD: (○)
[42] PUS: (+)
[43] AZS: (+)
[44] EAM: (+)
[45] GLD: (+)
[46] DVA: (+)

zit angeboten[47]. Optionale Parameter sind zwar entsprechend markiert, jedoch sind fast alle Parameter optional, so dass dies keinen großen Mehrwert darstellt[48]. Für bestimmte Datentypen sind einfache Regeln verfügbar, so z. B. die Länge von Zeichenketten oder die Anzahl von Nachkommastellen bei Zahlen. Komplexere Regeln sind hier jedoch nicht möglich[49]. Icons werden für die Services nicht angeboten[50]. Bei jedem Funktionsaufruf können Sprache und Zeitzone mit angegeben werden, so dass die Ausgabe entsprechend angepasst wird[51].

Insgesamt lässt sich sagen, dass OpenERP eine solide API bereitstellt und damit gerade im Vergleich zu anderen OpenSource-Lösungen weit vorne liegt. Was Umfang und Dokumentation angeht ist es jedoch nicht vergleichbar mit den großen kommerziellen Lösungen wie zum Beispiel von SAP. Für kleine und mittlere Unternehmen ist es sicherlich interessant den Einsatz einer Lösung wie OpenERP in Erwägung zu ziehen, da zum einen keine Anschaffungskosten anfallen und es sich zum anderen durch die solide API relativ gut in eine serviceorientierte Architektur einbinden lässt. Man sollte bei der Benutzung auf jeden Fall bedenken, dass hier keine Versionierung vorhanden ist und somit auch keine Gewährleistung, was die Stabilität der API und damit die Zukunftssicherheit angeht.

Apache Ofbiz Apache Ofbiz [Apa13] ist ein weiteres Open-Source-ERP-System. Hier wird es in Version 11.04.02 betrachtet. Die API basiert technisch ursprünglich auf einer eigenen, proprietären Technologie, bei der die Services in einem eigenen XML-Format beschrieben werden. Dabei werden so genannte „Simple-Services“ und „Java-Services“ unterschieden. Bei ersteren werden die Services komplett in der XML-Datei beschrieben, bei letzteren befindet sich dort mit einem Java-Klassennamen sowie einem Methodennamen lediglich der Verweis auf einen Service. Diese proprietären Dienste werden in der aktuellen Ofbiz-Version nicht mehr direkt angeboten, sondern werden über zwei alternative Service-Technologien angeboten. Diese Technologien bilden dabei quasi einen „Wrapper“ um die alte Technologie. Im einzelnen sind dies XML-RPC sowie durch WSDL-Dokumente beschriebene SOAP-Web-Services. Die WSDL-Definitionen werden hier jedoch nicht wie gewohnt zur Beschreibung der Service-Schnittstellen genutzt. Diese Information bleibt in den proprietären XML-Dateien. Außerdem ist sie noch in den Dokumentations-Tags der WSDL vorhanden. Ansonsten sieht in der WSDL jeder Service

[47] WVA: (+)
[48] DOP: (∘)
[49] VVA: (+)
[50] GIS: (−)
[51] ISN: (+)

gleich aus. Sowohl für die Eingabedaten als auch für die Ausgabedaten existiert jeweils eine Map mit beliebigem Inhalt. Eine Generierung von Java-Stubs mit z. B. dem Tool `wsimport` von JAX-WS liefert hier also keine wirkliche Hilfestellung.

Eine einfache Registry ist dadurch vorhanden, dass eine Liste der URLs der WSDL-Dokumente aller verfügbaren Services unter einer festen URL verfügbar gemacht wird[52]. Die Definitionen der Services sind technisch korrekt. Dies wird durch die Entwickler allein schon dadurch sehr regelmäßig getestet, dass diese Services auch intern in der Software benutzt werden[53]. Die Abdeckung der CRUD-Funktionen für Business-Objekte ist sehr gut. Create, Read und Update wird grundsätzlich angeboten. Delete-Services gibt es nur für Personen (Customer, Employee, Supplier)[54]. Eingabedaten werden vor der Ausführung überprüft[55], eine Versionierung der API existiert nicht[56].

Die Namen der Services und Objekte sind zwar einigermaßen intuitiv, jedoch nicht immer konsistent gewählt. Zum Beispiel wird ein Kunde mal über „customer“, mal über „person“ und an einer anderen Stelle als „party“ angesprochen[57]. Die API ist insgesamt nicht durch Kategorien strukturiert und die einzelnen Services sind oft so klein, dass viele Funktionsaufrufe nötig sind, um an zusammengehörige Daten zu gelangen. So gibt es zum Beispiel beim Kundenobjekt separate Funktionen zum Lesen der Adresse, der E-Mail und der Telefonnummer[58]. Eine Dokumentation der API ist kaum vorhanden, so dass es oft schwierig ist, die korrekten Service-Namen zu identifizieren. Auch sind nur sehr wenige Beispiele für die Benutzung der API vorhanden[59].

Durch die Benutzung von SOAP/WSDL bzw. XML-RPC werden grundsätzlich plattformunabhängige Technologien verwendet[60]. Dabei ist jedoch immer zu beachten, dass diese Technologien hier nicht so eingesetzt werden, wie sie gedacht sind, sondern lediglich als Transportschicht für das proprietäre Protokoll. Zur Autorisierung der Benutzer werden rollenbasierte ACLs[61] eingesetzt. Optional kann für die Übertragung HTTPS eingesetzt werden[62]. Die Authentifikation erfolgt über die Angabe von

[52] RSD: (+)
[53] KVS: (+)
[54] VZB: (+)
[55] EDV: (+)
[56] SSD: (−)
[57] IKB: (○)
[58] KES: (−)
[59] KVD: (−)
[60] PUS: (+)
[61] ACL: Access Control List
[62] AZS: (+)

Benutzername und Passwort bei jedem Aufruf[63].

Die API bietet weder Dokumentation[64] noch Auswahlwerte[65], noch Validierungsregeln[66] oder Icons[67] an. Optionale Parameter sind lediglich im Dokumentations-Tag der WSDL entsprechend gekennzeichnet, wobei außerdem fast alle Parameter optional sind[68]. Man kann den Services wie bei OpenERP eine Zeitzone und eine „Locale“ übergeben, jedoch wird dadurch keine Übersetzung der Elemente der API selbst angeboten[69].

Insgesamt ist die API von Apache Ofbiz im Vergleich zu der von OpenERP recht kritisch zu betrachten, besonders aufgrund der eigenwilligen Verwendung der WSDLs. Eigentlich kann die Ofbiz-API nicht als echte Web-Service-API betrachtet werden. Der Einsatz von Standardtools zur Stub-Generierung schlägt hier fehl. Die kaum vorhandene Dokumentation und die fehlende Versionierung sind weitere Schwachpunkte.

Als Fazit zu Open-Source-ERP-APIs insgesamt lässt sich sagen, dass der Qualitätsstandard hier leider noch deutlich unter dem der kommerziellen Lösungen liegt. Es scheint so zu sein, dass die Open-Source-Communitys hier personell doch deutlich schlechter aufgestellt sind als die Entwicklungsabteilungen der großen Hersteller kommerzieller ERP-Systeme. Bei einer aufwändigen Arbeit wie der Pflege einer guten API samt Dokumentation macht sich dieses deutlich bemerkbar.

3.5.2 E-Commerce-APIs

Ähnlich große APIs wie die der ERP-Systeme bieten die bekannten E-Commerce-Plattformen Ebay und Amazon. Da sich diese ebenfalls in der Domäne „Business“ aufhalten, können sie auch der Kategorie „Business-APIs“ zugeordnet werden. Da beide Angebote eine ähnliche Funktionalität anbieten - man kann auf der einen Seite anbieten und verkaufen, auf der anderen Seite Produkte suchen und kaufen - und da beide Angebote von großen und sehr erfolgreichen Unternehmen im Bereich des E-Commerce stammen, ist anzunehmen, dass die APIs recht vergleichbar und evtl. sogar sehr ähnlich sind. Ob dem so ist wird im Folgenden untersucht.

[63] EAM: (+)
[64] DVA: (−)
[65] WVA: (−)
[66] VVA: (−)
[67] GIS: (−)
[68] DOP: (◦)
[69] ISN: (◦)

Amazon Marketplace API Für die Nutzung der Amazon Marketplace API[70] ist ein so genannter „Pro-Merchant-Account" erforderlich, ein kostenpflichtiger Account für Händler. Ein kostenloses Testen der API ist also nicht möglich. Technisch basiert die API auf dem REST-Prinzip, wobei hier XML über HTTP übertragen wird. Außerdem sind Client-Bibliotheken für Java, C# und PHP vorhanden. Eine Registry ist hier nicht vorhanden, Informationen über geforderte Eingabedaten sind lediglich den Dokumentationen im PDF-Format zu entnehmen und eine Versionierung der API ist nicht erkennbar. In den zentralen Punkten schneidet die API demnach recht schlecht ab. Durch eine konsistente Namensgebung und klare und einfache Strukturen ist die API immerhin von der Usability-Seite recht gut gemacht. Technisch gibt es nichts auszusetzen. Die Technologie ist plattformunabhängig und eine Verschlüsselung ist durch die Nutzung von HTTPS gegeben. Von den in Abschnitt 3.1.4 aufgezählten zusätzlichen Informationen werden gar keine angeboten. Insgesamt ist die API also noch an vielen Punkten verbesserungswürdig, besonders an den zentralen Anforderungen einer Business-API sowie den vielen zusätzlichen Informationen, die es vereinfachen würden, automatisiert mit der API zu arbeiten. Außerdem ist zu bemängeln, dass die API nicht kostenlos getestet werden kann.

Ebay Trading API Anders als bei Amazon ist für den Zugriff auf die Ebay Trading API[71] lediglich ein kostenloser Developer-Account notwendig. Es existiert eine ausführliche Dokumentation als PDF-Dateien sowie viele Hilfestellungen und Code-Beispiele. Zum Testen der API steht eine Sandbox zu Verfügung, wobei es sich um eine vollwertige Kopie des echten Ebay-Datenbestands handelt. Für Java, Python und .NET werden SDKs angeboten. Technisch setzt die Ebay-API auf durch WSDL beschriebene SOAP-Webservices. Die API von Ebay ist in mehrere „Sub-APIs" unterteilt, wobei jede durch eine WSDL beschrieben wird. Über die hier angebotenen Funktionen sind alle Aktionen möglich, die man auch über die Ebay Webseite durchführen kann. Lediglich das Anlegen neuer User ist eine Ausnahme. Die API besitzt eine Versionierung, wobei eine Abwärtskompatibilität von mindestens 18 Monaten gewährleistet wird. Zu jedem Release sind detaillierte Informationen online einsehbar. Die API ist insgesamt sehr gut zu benutzen. Die Namen sind intuitiv und konsistent gewählt, die Struktur ist klar und einfach und die Dokumentation sehr umfangreich, korrekt und auf dem neuesten Stand. Die verwendeten Technologien sind plattformunabhängig und sicher und die Authentifizierung geschieht über ein entsprechendes User-Token. Die API bietet mehrere Zusatzinformationen an. Neben der vorbildlichen

[70] https://developer.amazonservices.de

[71] http://developer.ebay.com/common/api/ und https://www.x.com/developers/ebay

Nutzung der Dokumentations-Tags in der WSDL werden auch Werte für alternative Auswahlen angeboten. Optionale Parameter sind entsprechend gekennzeichnet und Fehlermeldungen werden in der gewünschten Sprache zurückgegeben.

Insgesamt ist die API von Ebay also sehr positiv zu bewerten. Sie ist ähnlich vorbildlich gestaltet wie die meisten beschriebenen Cloud-Dienste. Besonders im Vergleich zur Amazon Marketplace API sticht die Ebay API positiv hervor. Die Vermutung, die APIs könnten recht ähnlich sein, konnte absolut widerlegt werden. Die APIs verfolgen grundsätzlich sehr verschiedene Ansätze. Außerdem schneidet die Amazon API in fast allen Punkten schlechter ab als die API von Ebay.

3.6 Entwicklungsleitfaden für Business-APIs

Nach der Untersuchung der diversen Business-APIs gelangt man als Fazit zu einer ganzen Sammlung von Ratschlägen, wie eine solche API im besten Falle aussehen sollte und auch welche Wege nicht eingeschlagen werden sollten. Aus diesen Ratschlägen wurde der nun folgende Entwicklungsleitfaden erarbeitet. Dieser ist sowohl anwendbar auf existierende Applikationen ohne API oder neue Applikationen, die eine API anbieten sollen.

Der wichtigste Grundgedanke ist zunächst, dass der Entwicklung der API eine entsprechende Bedeutung beigemessen wird. Die API ist heutzutage mindestens genauso wichtig wie die GUI. Auch bei der API müssen demnach Usability-Test (z. B. mit Think-Aloud-Protokollen) durchgeführt werden und es muss kontinuierlich versucht werden, den Status Quo zu verbessern.

Ein erster naiver Gedanke zur Bereitstellung einer API könnte sein, dass es mit heutigen Technologien wie JAX-WS sehr einfach möglich ist, einige Annotationen an die entsprechenden Methoden zu heften und den Rest (z. B. das WSDL-Dokument) durch automatische Generierung zu erzeugen. Dieser Weg wird ausdrücklich nicht empfohlen. Auf diese Weise besteht keine Chance, ein wohldefiniertes, einheitliches Konzept zu verfolgen. Stattdessen werden einfach die internen Methoden wie sie sind nach außen freigegeben. Eventuell ist es jedoch gewünscht, dass die öffentliche API ganz anders strukturiert ist. Besonders eine gewisse Einheitlichkeit und Konsistenz kann so nicht erreicht werden.

Besser als eine automatische WSDL-Generierung ist hier der Ansatz „Contract First“[72]. Dabei wird im Falle von SOAP-Web-Services zuerst das WSDL-Dokument und das XML-Schema erstellt und anschließend

[72] http://rphgoossens.wordpress.com/2011/02/20/developing-a-contract-first-jax-ws-webservice/

die Implementierung vorgenommen. Die Implementierung beginnt dabei mit einer automatische Generierung von Stubs auf Basis von WSDL und XML-Schema. Im Falle von Java und SOAP bietet sich hier zum Beispiel das JAX-WS-Maven-Plugin an[73]. Die Entwicklung von WSDL und XSD muss dabei nicht von Hand erfolgen. Hier gibt es heute gute Tool-Unterstützung, so dass auch dies Modell-getrieben (z. B. nach dem MDA-Konzept[74] der OMG [KWBE03]) durchgeführt werden kann [GSSO04, BBCT04, FP02].

Für die Implementierung bietet sich die Standardtechnologie SOAP-Web-Services an, da diese erprobt ist und sich allgemein durchgesetzt hat. Außerdem ist sie Plattform-unabhängig, was gerade in der Zeit des Cloud-Computing besonders wichtig ist. Es gibt auch keinen Grund mehr, sich eine proprietäre Lösung auszudenken. Bei proprietären Lösungen ist jeder Anwender zunächst dazu gezwungen, sich in diese spezielle Technologie neu einzuarbeiten. Dies widerspricht dem ITSy-Gedanken. Außerdem sollte man eine Technologie so nutzen, wie sie gedacht ist. Eine Zweckentfremdung wie bei Apache Ofbiz (siehe Abschnitt 3.5.1) ist nicht ratsam. Aus gleichen Gründen sollte auch für die Authentifizierung und Autorisierung eine Plattform-unabhängige Standardtechnologie gewählt werden. Bei der Nutzung von SOAP-Web-Services hat man durch WSDLs eine wohldefinierte Schnittstelle und damit nicht das Problem der Unterspezifikation, wie es oft bei REST-Schnittstellen der Fall ist (siehe Abschnitt 2.3). Es ist jedoch auch keine große Aufgabe, zusätzlich eine REST-basierte Schnittstelle mit der gleichen Struktur anzubieten. Dies hat Salesforce zum Beispiel vorgemacht. Diese kann deutlich einfacher als SOAP von mobilen Apps und HTML-Webseiten aus benutzt werden.

Die API und ihre Implementierung sollte ein eigenes technisches Projekt, also ein eigenständiges Modul, sein. So kann man es unabhängig vom eigentlichen Produkt, also dem Service-Lieferanten weiterentwickeln und versionieren.

Nachdem man sich für eine Technologie entschieden hat und sich darüber im klaren geworden ist, dass diesem Projekt einen hinreichenden Stellenwert beizumessen ist, muss nun ein fachliches Konzept angefertigt werden. Dazu gehört eine intensive Anforderungsanalyse. Eine gute API muss immer zum entsprechenden Projekt passen. Es existiert nicht das eine Patentrezept, nach dem alle APIs gleich aufgebaut sein sollten. Für die Aufstellung dieses Konzeptes müssen diverse Fragen beantwortet werden:

- Welche Services (also Funktionen) werden benötigt? Diese Frage

[73] http://jax-ws-commons.java.net/jaxws-maven-plugin/

[74] MDA: Model-driven Architecture

kann am besten durch User-Stories beantwortet werden. User-Stories können wiederum durch Geschäftsprozesse definiert werden (zum Beispiel in BPMN oder im jABC). Wichtig ist, dass man eine vollständige API erhält, die wirklich alles nötige erlaubt.

- Wie groß ist die API? Die Anzahl der Services hat entscheidenden Einfluss auf die Art und Weise, wie die API strukturiert wird.
- Wie kann die API strukturiert werden? Gibt es Kategorien? Wenn ja, gibt es hierarchische Kategorien? Kann man die Funktionen zu Business-Objekten zusammenfassen? Sind die Business-Objekte relativ gleichartig oder sehr verschieden? Kann man also die gleichen Operationen (z. B. CRUD) auf alle Business-Objekte anwenden? Es muss hier entschieden werden, ob der algebraische Ansatz gewählt wird oder der Funktionen-zentrische mit individuellen Operationen.
- Wie wird die Versionierung vorgenommen? Für die ganze API? Pro Kategorie auf oberster Ebene? Pro Business-Objekt? Pro Funktion? Wichtig ist dabei auch die Möglichkeit, Services als veraltet (also als „deprecated“) zu markieren. Dabei muss dann auch immer angegeben werden, welcher Service stattdessen benutzt werden soll.
- Wie werden die WSDL-Dokumente aufgeteilt? Kommt die gesamte API in eine WSDL? Gibt es eine WSDL pro Kategorie oder sogar eine WSDL pro Business-Objekt oder eine WSDL pro Service? Sind vielleicht sogar mehrere WSDLs pro Service nötig, für die verschiedene Aspekte jeweils eine?
- Wie wird die Registry umgesetzt? Bei nur einer WSDL ist diese nicht nötig. Ansonsten könnte man über UDDI nachdenken. Dieses hat sich jedoch in der Geschichte als zu komplex dargestellt und wird daher kaum eingesetzt. Die proprietäre Microsoft-Lösung DICSO ist auch keine gute Wahl, da diese zum einen kaum Features bietet, zum anderen außerhalb der Microsoft-Welt nicht bekannt ist. Eine gute Idee können **Meta-Services** sein, wie sie in [Sch10]) beschrieben werden. Dies sind Services, die Informationen über Services liefern können. Rainer Schmidt [Sch10] erweitert dies sogar noch dahingehend, dass die Services auch modifizieren können, also neue Services zu deployen, alte zu entfernen etc. Meta-Services sind also Services, die mit Service-Informationen hantieren[75]. Ein Meta-Service kann zum Beispiel die Liste aller Services zurückgeben. Meta-Services können auch weitere Informationen liefern (also alles aus der Kategorie „zusätzliche Informationen“, wie Dokumentation, internationalisierte Strings, Regeln, etc.). Manchmal ist es

[75] Nicht gemeint sind hier Meta-Services, wie zum Beispiel bei Meta-Suchmaschinen oder ähnlichen Meta-Diensten (wie zum Beispiel in [LKRP+08] oder [LWC07]), wo mehrere gleichartige Services zu einem größeren oder abstrakterem zusammengefasst werden.

hier alternativ jedoch einfacher, Zusatzinformationen strukturiert und einheitlich in den `documentation`-Tags der WSDL zu platzieren. Auch die SAP-BAPI enthält Beispiele für Meta-Services, zum Beispiel für die F1- und F4-Hilfen. In beiden Fällen sind dies Services, die den Namen von anderen Services übergeben bekommen und dementsprechend Informationen über diese Services zurückliefern. Meta-Services stellen also eine Art „Selbstbetrachtung" des Systems dar, vergleichbar mit dem Konzept der „Reflection" [FFI04] in Programmiersprachen wie Java.

- Wie wird die Authentifizierung durchgeführt? Sollen bei jedem Aufruf erneut Benutzername und Passwort übertragen werden oder soll eine Sitzung mit einem entsprechenden Sitzungsschlüssel erzeugt werden, welcher immer wieder übertragen wird? Beide Ansätze besitzen ihre Vor- und Nachteile. Der Aufbau einer Session ist evtl. technisch komplexer und ist ein technisches Detail, welches oft trotzdem im Prozessdiagramm auftaucht. Andererseits ist dieser Ansatz sehr elegant und effizient. Die Authentifizierung für jeden einzelnen Service-Aufruf ist insgesamt einfacher in der Benutzung, jedoch auch weniger effizient.

Grundsätzlich gilt, dass Dokumentation extrem wichtig ist und ausnahmslos alles dokumentiert werden muss. Dabei ist auch wichtig, dass die vorhandene Dokumentation auch an den entsprechenden Stellen zugänglich ist. Die Dokumentation gehört also nicht nur auf eine Webseite oder in ein PDF, sondern in die WSDL. Web-Service-Frameworks sollten dann diese Dokumentation auch als Dokumentation in ihr Generat übernehmen, z. B. als JavaDoc. Dokumentation darf an keiner Stelle verloren gehen.

Man sollte ein Konzept von Business-Rules haben, welche auch als Validierungsregeln gelten. Diese sollten wohl-strukturiert hinterlegt werden. Business-Rule-Engines wie z. B. JBoss Drools[76] können hier eine gute Wahl sein. Wenn ein Service-Aufruf gegen eine Regel verstößt, muss dies explizit gemeldet werden. In [SG06] wird anschaulich dargelegt, wie Business-Rules einem Unternehmen helfen können, eine agile IT aufzubauen. Dabei vervollständigen sich Business-Prozesse und Business-Rules gegenseitig. Beide zusammen beschreiben sehr gut das Gesamtbild eines Unternehmens.

Es kann für den Anbieter einer API sehr sinnvoll sein, zusätzlich auch entsprechende Client-Bibliotheken in verschiedenen Programmiersprachen anzubieten. Damit entfällt schon sehr viel Arbeit für denjenigen, der diese Dienste benutzen möchte. Google ist hierfür ein sehr gutes Beispiel. Der Anbieter selbst kennt seine API und dessen technische Eigenarten

[76] http://www.jboss.org/drools/

am besten und ist selbst am besten in der Lage, diese zu verstecken. Hat die API eine überschaubare Größe mit einer relativ festen Zahl von Services, so kann dies gut gemacht werden. Ist die API größer und möchte man flexibler bei Bedarf konkrete Services einbinden, so bietet sich eher ein Ansatz wie bei Salesforce (oder bei SAP-BAPI und JCo) an, die zwar eine Client-Bibliothek anbieten, welche jedoch nur technisch ausgerichtet ist.

KAPITEL 4

Service-Integration in BPMS

Die Integration von Services geschieht je nach verwendeter Prozessbeschreibungssprache und auch abhängig vom konkret verwendeten Tool auf sehr unterschiedliche Wege. Im Folgenden wird zunächst beschrieben, was hier die Spezifikation von BPMN 2.0 beschreibt, was im Zuge des aktuellen Hypes viele heute wohl als neue „Lingua Franca“ des BPM sehen würden. Anschließend wird eine Kategorisierung der grundsätzlichen Vorgehensweisen zur Service-Integration vorgenommen, wonach auf jede einzelne Kategorie eingegangen wird. Zuletzt wird noch auf die Struktur, die Erstellung und die Organisation so genannter „Business-Aktivitäten“ eingegangen, deren Einsatz die heute fortschrittlichste Art der Service-Integration darstellt.

4.1 Service-Integration in BPMN 2.0

Im Unterschied zur ersten Version hat BPMN in seiner Version 2.0 den Anspruch, eine ausführbare Geschäftsprozessnotation zu sein. Gleichzeitig ist BPMN jedoch neutral gegenüber konkreten Technologien zur Bereitstellung, Nutzung und Implementierung von Services. Dies ist ein Spagat, der nur schwerlich zu realisieren ist. Wenn in der Spezifikation nicht über Technologien gesprochen werden darf, bleibt es den Tool-Herstellern überlassen, wie genau Services eingebunden werden sollen. Das bedeutet wiederum, dass jeder Hersteller hier potentiell einen anderen Weg wählt und ein Austausch ausführbarer Prozessbeschreibungen zwischen Tools verschiedener Hersteller nicht möglich ist.

Die Technologieunabhängigkeit von BPMN macht sich hier darin bemerkbar, dass nur sehr wenig zum Thema Service-Integration ausgesagt wird. Prozesselemente, die einen Vorgang enthalten (also irgendetwas, was eine bestimmte Zeit in Anspruch nimmt), heißen in BPMN „Aktivitäten“. Dieser Begriff wird im Folgenden auch allgemein (d. h. auch in anderen Prozessbeschreibungssprachen als BPMN) verwendet[1]. Eine Aktivität kann in BPMN eine Aufgabe (engl. „Task“) oder ein Teilprozess (engl. „sub-process“) sein. Eine Aufgabe ist wiederum zum Beispiel eine Benutzeraufgabe (engl. „User-Task“), eine Skript-Aufgabe „Script-Task“ oder eine Service-Aufgabe (engl. „Service-Task“). Da sich die deutschen Bezeichnungen im allgemeinen Sprachgebrauch in der Community nicht durchgesetzt haben, werden im Folgenden die englischen Varianten verwendet. Wie der Name suggeriert, sind Service-Tasks die für die Service-Integration vorgesehenen Elemente. Deren Struktur ist sehr einfach gehalten. Neben der Notation[2] wird lediglich definiert, dass ein Service-Task aus folgenden Elementen besteht:

- `name`: Der Name
- `inMessageRef`: Die Eingabedaten
- `outMessageRef`: Die Rückgabedaten
- `errorRef`: Informationen über mögliche Fehler
- `implementationRef`: Die Angabe der verwendeten Service-Technologie[3].

Die Ausführungssemantik der BPMN-Service-Tasks wird in einem recht kurzen Abschnitt erklärt, der folgendes aussagt:

> „Service Task: Upon activation, the data in the inMessage of the Operation is assigned from the data in the Data Input of the Service Task the Operation is invoked. On completion of the service, the data in the Data Output of the Service Task is assigned from the data in the outMessage of the Operation, and the Service Task completes. If the invoked service returns a fault, that fault is treated as interrupting error, and the Activity fails.“

Hier wird also lediglich definiert, dass es Ein- und Ausgabedaten gibt und dass ein Fehler bei der Ausführung auftreten kann, der behandelt werden sollte. Diese Definition lässt großen Spielraum für BPMS-Hersteller, so dass komplett proprietäre Lösungen die Folge sind.

[1] Zum Beispiel auch in den Kapiteln 5 und 6, in denen es um die Service-Integration in jABC respektive anderen BPMS geht.
[2] Service-Tasks werden als abgerundete Rechtecke mit zwei überlappenden Zahnrädern dargestellt.
[3] Hier sind durch WSDL beschriebene SOAP-Webservices (konkret: `##webService`) der vorgegebene Standard.

4.2 Kategorisierung

Es können drei grundsätzlich unterschiedliche Arten identifiziert werden, wie Services in ausführbare Prozesse integriert werden. Für jede dieser drei Arten soll eine Kategorie definiert werden:

1. Skript-Aktivitäten
2. Technische Service-Aktivitäten
3. (Domänen-spezifische) Business-Aktivitäten

Im Folgenden werden die einzelnen Herangehensweisen definiert, näher beschrieben und die jeweiligen Vor- und Nachteile in bestimmten Bereichen aufgezeigt.

4.3 Skript-Aktivitäten

Skript-Aktivitäten erlauben es, Code einer Programmiersprache[4] direkt am entsprechenden Knoten im Prozessmodell zu annotieren. Code wird also direkt in das Prozessmodell eingefügt, Modellierung und Implementierung werden vermischt. Dieser Code wird dann während der Prozessausführung bei Erreichen der Skript-Aktivität ausgeführt. Dieses Feature wird üblicherweise für das „Rapid Prototyping“[5] oder für sehr kleine Aufgaben genutzt. Alle anderen Anwendungen stellen eine Art von „Missbrauch“ dar. Der Code, der nur direkt in dieser Aktivitäten-Instanz existiert, kann nur durch „Copy-and-Paste“ wiederverwendet werden, was schnell zu schwer wartbaren Lösungen führt. Dadurch, dass im Prinzip jeder Code in einer Skript-Aktivität erlaubt ist, ist es möglich, dass hier externe Services aufgerufen werden.

Die Schwäche der mangelnden Wiederverwendung ist nicht der einzige Nachteil. Zusätzlich wird hier ein fundamentales Konzept des Software-Engineerings verletzt: „Separation of concerns“ (Trennung der Angelegenheiten). Unterschiedliche Aspekte sollten an getrennten Orten behandelt werden, so dass ein möglichst modulares System entsteht. Nur so kann man Sichten auf ein Gesamtsystem gewinnen, die ein insgesamt komplexes System für einzelne Beteiligte verständlich darstellen. Hier ist es wichtig zu beachten, dass es im Bereich des Geschäftsprozessmanagements verschiedene Nutzergruppen mit sehr unterschiedlichen Fähigkeiten, Kenntnissen und Neigungen gibt. Der Business- oder Anwendungsexperte kennt die Prozess im Unternehmen oder im Markt, weiß

[4] oft einfache Skript-Sprachen wie z. B. JavaScript oder Groovy

[5] Die schnelle Entwicklung einer oft wenig getesteten und nur halb-fertigen aber lauffähigen Lösung, die als Basis zur Kommunikation mit dem Kunden dienen kann.

jedoch nicht, wie ein Service implementiert wird. Ein IT-Experte kennt wiederum den Prozess nicht, kann jedoch die Services implementieren, die vom Anwendungsexperten verlangt werden. Diese beiden Rollen müssen klar getrennt werden um Prozess-Design mit keinem oder geringem technischen Wissen zu ermöglichen. Skript-Aktivitäten sind in dieser Hinsicht sicherlich ungeeignet. Kein Anwendungsexperte möchte selbst Programmcode in die Prozesse einfügen.

Am Misserfolg von BPELJ [BGK+04], einer offiziellen Erweiterung für WS-BPEL 2.0 [OAS07], kann man gut erkennen, wo die Probleme des Ansatzes insgesamt liegen. Hier wurde es erlaubt, Java-Code direkt in den XML-Code von BPEL einzufügen, um möglichst flexibel beliebige Funktionalitäten aus BPEL heraus aufzurufen. Beide oben erwähnten Nachteile gelten hier, so dass es diese Spezifikation nie wirklich in die Praxis geschafft hat.

Howard Smith [Smi04], welcher ebenfalls auf die genannten Nachteile von BPELJ einging, wies darauf hin, dass eine klare Trennlinie zwischen dem Prozess und den zugrunde liegenden Services geben muss:

> „Business people understand that when their pure programming requirements exceed that of tools like BPMS (...) they ask IT developers to create or find additional code for them that can be included as participants in business processes.“

Skript-Aktivitäten sollten also nicht benutzt werden um Services *in* einen Prozess einzubinden. Stattdessen müssen sich Services *unter* dem Prozess befinden wobei die Implementierungen für den Prozess-Designer sogar komplett versteckt (also unsichtbar) sein müssen. Aus dem Beispiel BPELJ könnte man sogar so weit gehen und Skript-Aktivitäten ganz verbieten. In produktiv laufenden Prozessen ist deren Abwesenheit wohl der Wunschzustand.

Obwohl Skript-Aktivitäten nicht die erste Wahl für die Aufgabe der Service-Integration darstellen, ist es nichtsdestotrotz sinnvoll, diese in diesem Kontext zu betrachten. Zum einen werden sie in der Praxis zu diesem Zweck genutzt, zumindest für das Rapid Prototyping. Zum anderen ist es oft nicht ganz klar zu welcher Kategorie eine Aktivität gehört. Zum Beispiel kann eine Aktivität, die eine Expression[6] als Parameter enthält, schon als Skript-Aktivität gesehen werden, mit allen oben genannten Nachteilen. Dies gilt nicht nur für Expressions. Alle komplexen, technischen Zeichenketten, die auf irgendeine Art und Weise ausgewertet werden (z. B. SQL) können als direkte Parameter von Aktivitäten problematisch sein.

[6] also einen zu interpretierenden Ausdruck

4.4 Technische Service-Aktivitäten

Vielen BPMS merkt man an, das bei ihrer Entwicklung hauptsächlich die dahinter liegenden Technologien im Fokus der Aufmerksamkeit standen. Dies führt dazu, dass die am häufigsten eingesetzte Technik zur Service-Integration sich auf die Technologie bezieht, mit der der Service angeboten wird. Ein gutes Beispiel dafür ist WS-BPEL. Hier gibt es die Aktivität `<invoke />` für den Aufruf eines durch WSDL definierten SOAP-Webservices. Bei der Benutzung dieser Aktivität (oder irgendetwas anderem in WS-BPEL) muss der Prozessmodellierer ein tiefes Verständnis für alle beteiligten Technologien (wie WSDL, SOAP, XSD, manchmal auch XSLT) aufweisen. Ein Prozessmodellierer denkt typischerweise in Schritten wie „Führe Preiskalkulation durch“ und ein Wechsel im Denken hin zu Schritten wie „Rufe den Webservice auf, welcher die Preiskalkulation ausführt“ kann eine gewaltige Hürde darstellen, besonders wenn auch noch viele andere Technologien existieren.

WS-BPEL ist nicht die einzige Prozessbeschreibungssprache mit solchen „technischen Service-Aktivitäten“ (wie sie im Folgenden weiter genannt werden sollen). Fast alle Sprachen und Systeme bieten Aktivitäten wie zum Beispiel eine „Java-Aktivität“ an um Java-Methoden aufzurufen, eine „SQL-Aktivität“ um SQL-Skripte[7] auszuführen oder eine „Webservice-Aktivität“, welche sich genau wie das `<invoke />` in WS-BPEL verhält.

Technische Service-Aktivitäten sind im Allgemeinen schon viel besser geeignet für die Service-Integration als Skript-Aktivitäten. Sie sind spezialisierter und erfordern normalerweise nicht die Kenntnis einer Programmier- oder Skriptsprache. Nichtsdestotrotz werden hier Geschäftsprozess und Service-Technologien immer noch nicht hinreichend voneinander getrennt.

4.5 Business-Aktivitäten

Es dem Business-Experten zu ermöglichen ausschließlich in dem ihm vertrauten Business-Vokabular zu denken, kann durch die Einführung einer entsprechenden DSL („Domain-specific Language“) [MHS05, VDKV00] erreicht werden. Es sollte klar sein, dass Prozesssprachen, die hauptsächlich auf Aktivitäten setzen wie sie in den Abschnitten 4.3 und 4.4 beschrieben wurden, keine Business-DSL darstellen können. Van Deursen et al. [VDKV00] definieren eine DSL folgendermaßen:

> „A domain-specific language (DSL) is a programming language or executable specification language that offers, through

[7]Diese Aktivitäten könnte man auch den Skript-Aktivitäten zuordnen.

> appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.“

Folgende Punkte sind demnach zu erfüllen:

1. Programmier- oder Spezifikationssprache
2. Ausführbarkeit
3. Fokus auf ein bestimmtes Anwendungsgebiet („Problem Domain“)
4. Beschränkung der Mächtigkeit („restricted“)

Punkt 1 ist für alle Prozesssprachen erfüllt, da es sich um Spezifikationssprachen für Prozesse handelt, Punkt 2 besagt, dass wir uns hier mit ausführbaren Sprachen auseinandersetzen. Das bedeutet, dass in Sprachen wie EPK, BPMN 1.x oder die Aktivitätsdiagramme der UML herausfallen. Diese Sprachen sind reine Notationen ohne konkrete Ausführungssemantik. Punkt 3 verlangt, dass man ein bestimmtes Anwendungsgebiet im Auge hat. Für eine DSL mit dem Anwendungsgebiet „Business“ sind also Aktivitäten nötig, die Geschäftstätigkeiten im engeren Sinne repräsentieren. Die Aktivitäten sind die Bausteine der Prozesse, entsprechen also den Befehlen einer Programmiersprache. Sie machen die Anwendbarkeit und Mächtigkeit der Sprache aus. Wenn man also eine Prozesssprache bewertet und prüft, ob es sich hier um eine DSL handelt, so muss man die Sammlung der Aktivitäten betrachten. Punkt 4 besagt, dass man mit dieser Sprache nicht jedes Problem lösen können muss. So ist es kaum sinnvoll, mit BPEL einen Sortieralgorithmus zu implementieren. Diese bewusste Einschränkung ist auch ein gutes Argument gegen den Einsatz von Skript-Aktivitäten.

Der besonders wichtige Punkt 3 fordert für eine Business-DSL also konkret, dass eine Aktivität eine Geschäftsaktivität, also einen Service, repräsentiert. Jede solche Aktivität bezieht sich also auf genau einen Service. Es kann auch mehrere Aktivitäten pro Service geben, wenn der Service jeweils unterschiedlich genutzt wird oder eine Aktivität, die wiederum einen gesamten Prozess aufruft, der darunterliegende Services orchestriert.

Der große Unterschied zu den technischen Service-Aktivitäten ist, dass eine feste Relation zwischen den Aktivitäten und den dazugehörigen Services existiert. Man hat als Prozessmodellierer nun also einen Werkzeugkasten mit Aktivitäten wie „Auftrag anlegen“ oder „Mitarbeiter einstellen“ und nicht mehr „Webservice aufrufen“ oder „Datenbank-Abfrage ausführen“. Solche Aktivitäten werden im folgenden „Business-Aktivitäten“ genannt. Ebenso zutreffend ist der Begriff „Domänen-spezifische Aktivitäten“. Prozesssprachen, die also hauptsächlich die Verwendung solcher Aktivitäten vorsehen, können als Business-DSL bezeichnet werden. So

können diese Aktivitäten und damit auch die Prozesssprachen ohne technisches Wissen benutzt werden. Außerdem sind diese Aktivitäten sehr gut wiederverwendbar.

Business-Aktivitäten stellen die fortschrittlichste Technik zur Integration von Services dar und deren Benutzung wird hiermit ausdrücklich empfohlen. Im Folgenden wird die Struktur, deren Erstellung und die Organisation von Business-Aktivitäten beschrieben.

4.5.1 Struktur

Eine Business-Aktivität stellt das Verbindungsglied zwischen Prozess und Service dar. Die technische Ansteuerung des Service muss daher Teil der Aktivität sein, jedoch für den Prozessmodellierer versteckt und somit unsichtbar. Er sieht nur die Definition der Aktivität. Wenn man also die grobe Struktur einer Business-Aktivität betrachtet, so hat man zwei Teile:

1. Definition
2. Implementierung

Die Implementierung besteht aus Programmcode, z. B. in Java. Dieser fällt natürlich nicht vom Himmel und muss irgendwie erstellt werden. Darauf wird in Abschnitt 4.5.2 näher eingegangen.

Die Definition besteht aus einer ganzen Reihe von Elementen:

1. Name / ID
2. Parameter
3. Dokumentation
4. Ausgänge als beschriftete, ausgehende Kanten (optional)
5. Icon (optional)
6. Benutzungsregeln (optional)

Die mit „optional“ gekennzeichneten Bestandteile sind nicht unbedingt notwendig um als Business-Aktivität zu gelten, erhöhen jedoch die Qualität der Aktivität enorm.

Die beschriebene Struktur der Business-Aktivitäten entspricht exakt der in Abschnitt 2.6 SIB-Struktur. SIBs sind also ein perfektes Mittel für die Umsetzung von Business-Aktivitäten. Dabei ist zu beachten, dass ein SIB nicht automatisch eine Business-Aktivität ist. Nur wenn beim Entwurf der SIBs die richtige Granularität gewählt wurde, das SIB alle oben genannten Kriterien erfüllt um Teil einer Business-DSL zu sein, und

das SIB hinreichend dokumentiert ist, so ist ein SIB auch eine Business-Aktivität.

Der Name (oder eine ID) identifiziert die Aktivität und damit den Service. Hierunter ist die Aktivität in einer Sammlung auffindbar und mit diesem Namen kann der Prozessmodellierer die Aktivität sowohl im Modellierungstool als auch in der Kommunikation mit den Entwicklern eindeutig benennen.

Die Parameter diesen zur Anpassung der Aktivität an eine konkrete Nutzung. Erst so ist auch eine gewisse Wiederverwendbarkeit gegeben. Diese Parameter beinhalten auch die Schnittstellenbeschreibung für Ein- und Ausgaben. Hier wird also genau festgelegt, woher die Daten kommen und wo die Ergebnisse abgelegt werden sollen. Kommunikation zwischen Aktivitäten wird fast immer durch einen entsprechenden Ausführungs-Kontext bewerkstelligt, welcher die Laufzeitdaten enthält und als „Shared Memory" für die Ausführungsumgebung dient.

Dokumentation ist für Business-Aktivitäten von hoher Bedeutung. Nur so können Prozessmodellierer verstehen, welche Funktionalität die Aktivität und damit der Service bereitstellt. Auch genaue Informationen über mögliche Ausgänge der Ausführung und die Parameter sind unerlässlich. Bei letzterem zählen auch genaue Darstellungen der Daten (samt Datentypen), die von SIB konsumiert und/oder produziert werden dazu. Gibt es weitere Seiteneffekte durch die Ausführung der Aktivität, so sind diese ebenfalls in der Dokumentation zu erwähnen und zu erklären.

Die Definition der Ausgänge in der Business-Aktivität selbst bietet einen entscheidenden Vorteil gegenüber einer freien Benennung der Kanten. So hat eine Aktion sehr oft verschiedene mögliche Ausgänge, die möglichst alle betrachtet werden sollten. Werden diese möglichen Ausgänge explizit mit der Aktivität definiert, so trägt dies zum einen dazu bei, die Aktivität besser zu verstehen und zum anderen, dass kein Ausgang unabsichtlich unberücksichtigt bleibt. Durch die verschiedenen Ausgänge ist jede Business-Aktivität gleichzeitig ein potentieller XOR-Split, also ein Entscheidungspunkt, an dem entschieden wird, welche ausgehende Kante gewählt wird. Dies führt zu einer sehr kompakten Prozessdarstellung, da keine zusätzliche XOR-Aktivität[8] benötigt wird. Wie oben schon erwähnt, ist die Definition der Ausgänge in der Aktivität selbst keine unbedingte Voraussetzung um als Business-Aktivität zu gelten. Tatsächlich verzichten die meisten Sprachen auf dieses Feature und nutzen eher zusätzliche Gateways für diesen Zweck.

Um die Benutzung einer Business-Aktivität noch komfortabler zu gestalten, bieten es manche Tools an, ihnen eigene graphische Icons zuzuweisen, so dass zum einen der modellierte Prozess leicht zu verstehen ist

[8] wie z. B. ein BPMN-XOR-Gateway

und zum anderen die Aktivität in einer Aktivitätensammlung (in Listen- oder Baumdarstellung) schnell auffindbar ist.

Letztlich kann die fehlerfreie Benutzung von Business-Aktivitäten noch durch die Definition von entsprechenden Regeln vereinfacht werden. Diese Regeln gelten lokal für die jeweilige Aktivität und höchstens noch für das direkte Umfeld, also z. B. für die ausgehenden, beschrifteten Kanten. Solche Regeln können Wertebereiche für Parameter sein oder in aufwendigeren Fällen auch spezifische Regeln für die richtige Benutzung, wie z. B. dass eine Aktivität nicht die letzte in einem Prozess sein darf.

4.5.2 Erstellung

Anders als Skript-Aktivitäten oder technische Service-Aktivitäten, die jeweils quasi direkt verwendet werden können, müssen Business-Aktivitäten für jeden Service einzeln erst einmal erstellt werden. Nicht der Definitionsteil und auch ganz besonders nicht der Implementierungsteil „fallen einfach vom Himmel“. Dies bedeutet also immer einen gewissen initialen Aufwand, der sich später auszahlt, wenn komplexe Prozesse auf eine agile Art und Weise erstellt werden. Service-Integration bedeutet grundsätzlich einen gewissen technischen Aufwand, den man nicht „wegzaubern“ kann. Was man machen kann und auch sollte ist, den Aufwand zum einen möglichst zu minimieren und dann zum anderen den übrig gebliebenen Aufwand möglichst geschickt auf die richtigen Personen verteilen. Jede beteiligte Person oder Rolle sollte dabei genau ihren Kenntnissen und Fähigkeiten entsprechend eingesetzt werden, die Kommunikation zwischen den Rollen sollte unterstützt und vereinfacht werden und es ist auch ganz besonders gewollt, dass alle beteiligten Rollen immer wieder miteinander kommunizieren um das gewünschte Ergebnis zu erzielen. Im Folgenden wird beschrieben, welche möglichen Wege existieren Business-Aktivitäten zu erstellen und wie erreicht werden kann, dass dieser Aufwand möglichst gut minimiert und organisiert wird.

Die naheliegendste Möglichkeit der Erstellung ist die manuelle Implementierung. In Fällen, in denen nur wenige Services eingebunden werden müssen, kann dies eine angebrachte Lösung sein. Es funktioniert direkt und kein weiterer Aufwand muss z. B. in Technologien zur Code-Generierung investiert werden.

Ein ausgefeilteres Verfahren ist die automatische Generierung des Definitionsteils basierend auf Daten, die in einer graphischen Benutzeroberfläche eingetragen werden. In einem recht einfachen Formular können dabei Schnittstellendaten (Eingabe, Ausgabe, ausgehende Kanten) sowie Metadaten (Dokumentation, Icon, Benutzungsregeln) für die Aktivität angegeben werden. Durch eine einfache, strukturierte Benutzeroberflä-

che wird hier erreicht, dass der Benutzer zum einen davor bewahrt wird, kein Element zu vergessen und zum anderen überhaupt erst befähigt wird, eine solche Aktivität zu erstellen ohne Kenntnisse einer konkreten Definitionssprachensyntax zu besitzen. Das einzige, was noch von einem Entwickler hinzugefügt werden muss, ist der Implementierungsteil der Aktivität, also der Code, der den Service aufruft. Die Generierung des Definitionsteils kann auch als „Stub-Generierung" bezeichnet werden, da hier nur ein Grundgerüst erzeugt wird, welches noch durch den eigentlichen Ausführungscode ergänzt werden muss. Dies unterscheidet sich von einer vollständigen Generierung, bei der auch der Implementierungsteil automatisch erzeugt wird.

Sehr vorteilhaft ist es, wenn die zu integrierenden Services irgendeine Art einer Schnittstellendefinition besitzen. Dies kann zum Beispiel ein WSDL-Dokument sein. Diese Definition kann genutzt werden, um vollautomatisch komplette Business-Aktivitäten (also Definitions- und Implementierungsteil) zu generieren. Die Generate sind in dem Fall also direkt ausführbar. In diesem Fall konnten also die Vorteile zweier Ansätze vereint werden, die *Einfachheit* der Business-Aktivitäten und die *direkte Verfügbarkeit* der technischen Service-Aktivitäten. Es muss dabei jedoch erwähnt werden, dass dieser Ansatz nur bei klaren Eins-zu-Eins-Beziehungen zwischen Aktivitäten und Services funktioniert. Die Aktivität ist also strukturell identisch mit dem Service. Dies bedeutet, dass möglicherweise komplexe Serviceschnittstellen zu ebenso komplexen Aktivitäten führen, was oft so nicht gewollt ist.

Eine halbautomatische Generierung mit Hilfe einer Wizard-ähnlichen Benutzeroberfläche[9] kann diese Probleme beheben: Hier kann der Generierungsvorgang durch Benutzerinteraktion beeinflusst werden, indem zum Beispiel die benötigten Parameter ausgewählt werden, beliebige Elemente umbenannt werden oder die Struktur verändert wird.

Technologien zur Code-Generierung für Business-Aktivitäten funktionieren am besten, wenn eine verhältnismäßig große Sammlung gleichartiger Services existiert. Dies ist zum Beispiel der Fall bei den APIs von ERP-Systemen. Hier kann es sich auszahlen, einen speziellen Code-Generator für jede Service-Sammlung zu erstellen, sodass der spezielle Charakter eines jeden Systems behandelt werden kann. So ist es möglich, dass selbst sehr verschieden umgesetzte APIs zu recht einheitlichen Sammlungen von Business-Aktivitäten führen. Außerdem müssen bestimmte technische Hürden, wie zum Beispiel ein kompliziertes Authentifizierungsverfahren, nur einmal genommen werden, und zwar bei der Erstellung des spezifischen Code-Generators, und nicht für jede Business-Aktivität erneut. Bei der Variante jeweils einen Code-Generator pro Technologie

[9]manchmal auch „Assistent" genannt, siehe auch Abschnitt 5.3.2

(z. B. WSDL/SOAP) einzusetzen entsteht zwar weniger Aufwand für die Erstellung der Generatoren, die spezifischen Eigenarten der Systeme werden jedoch nicht beachtet. Zwei Service-Sammlungen, die zum Beispiel beide WSDL einsetzen, können sehr unterschiedlich realisiert sein (siehe Kapitel 3). Diese Unterschiede würden dann wie sie sind auf Ebene der Business-Aktivitäten sichtbar werden. Bei System-spezifischen Generatoren ist dies zum einen nicht der Fall, zum anderen ist hier die Chance wesentlich größer, dass die generierten Aktivitäten auch in allen Fällen wirklich fehlerfrei funktionieren, und das ohne manuelle Manipulation des generierten Codes. Praktische Erfahrungen haben gezeigt, dass technische „Tricks“ auf Seiten der API-Anbieter, also besonders kreative Nutzungen der Service-Technologien, sehr schnell dazu führen können, dass die automatische Generierung auf Basis einer Technologie nicht funktioniert. Dies kann zum Beispiel daran liegen, dass der Service-Provider die .NET-Technologie von Microsoft einsetzt und der Service-Consumer Java. Und selbst innerhalb der Java-Welt kann es Probleme geben, wenn ein Dienst zum Beispiel mit JAX-WS angeboten wird und mit Axis2 konsumiert werden soll.

Zusammenfassend können die Generierungstechnologien also in zwei Dimensionen kategorisiert werden:

- Bezieht sich der Generator auf eine bestimmte Technologie oder auf ein System?
- Erfolgt die Generierung halb- oder vollautomatisch?

Es existiert hier kein klarer Kandidat als grundsätzliche beste Wahl. Stattdessen hat jede Option ihre Vor- und Nachteile, wie sie oben beschrieben wurden.

4.5.3 Organisation

Im Idealfall hat ein Prozessmodellierer eine große Sammlung von Business-Aktivitäten zur Verfügung. Dies führt einerseits zu enormen Möglichkeiten bei der Modellierung und es kann auch die Neu-Generierung weiterer Aktivitäten überflüssig machen. Andererseits kann ein Modellierer nur die Elemente einsetzen, die ihm auch bekannt sind. Es ist also wichtig, den Modellierer bei der Beherrschung großer Aktivitätensammlungen zu unterstützen. Genau genommen muss ihm bei der Suche nach den passenden Aktivitäten geholfen werden.

Als eine sehr geeignete Form der Organisation von Aktivitäten haben sich so genannte Taxonomien [VR03] erwiesen. Hier werden die Aktivitäten nach semantischen Kriterien kategorisiert. Taxonomien sind hierarchisch, d. h. jede Kategorie kann wieder Kategorien beinhalten. Außerdem kann

jede Aktivität auch in mehreren Kategorien vorkommen, evtl. auch unter verschiedenen Bezeichnungen. In seltenen Fällen werden Taxonomien noch erweitert durch spezielle Beziehungen, die über die reine Klassifizierung hinausgehen. In dem Fall spricht man nicht mehr von einer „Taxonomie“ sondern von einer „Ontologie“. In der Praxis hat sich herausgestellt, dass Taxonomien einen sehr guten Kompromiss zur Organisation von Business-Aktivitäten darstellen, da sie sehr einfach zu verstehen, zu benutzen und zu pflegen sind. Eine Taxonomie von Business-Aktivitäten ist genau das, was als „Business-DSL“ angesehen werden kann.

Leider unterstützen die meisten BPMS keine Taxonomien, sondern oft nur eine einfache, nicht weiter unterteilte Liste von Aktivitäten. Manchmal kann diese Liste noch auf einer Ebene in Kategorien aufgeteilt werden. Selten hat man eine echte hierarchische Struktur, die frei nach semantischen Gesichtspunkten gestaltet werden kann. Statt der logischen, semantischen Position in einer Taxonomie hat man oft auch nur die physische Position der Aktivität (z. B. der lokale Pfad zur Definitionsdatei).

Natürlich ist der physische Ort, welcher auch als ein Aspekt einer Taxonomie gesehen werden kann, nicht unwichtig. Business-Aktivitäten können zum Beispiel im Idealfall auf einem zentralen Server in einem Netzwerk (z. B. dem Internet) bereitgestellt werden, so dass jede Installation eines Modellierungstools dieses „zentrale Repository“ nutzen kann.

KAPITEL 5

Service-Integration im jABC

Im Folgenden wird beschrieben, wie die Integration von Services im Kontext von XMDD (siehe Abschnitt 2.6) und genauer im entsprechenden Tool jABC durchgeführt wird. In Abschnitt 5.1 wird dargelegt, wie alle drei in Kapitel 4 beschriebenen Kategorien von Aktivitäten mit SIBs umgesetzt werden. Darauf folgt in Abschnitt 5.2 eine Beschreibung verschiedener SIB-Implementierungen, die manuell durchgeführt wurden. Abschnitt 5.3 befasst sich mit der automatischen Generierung von SIBs, wobei hier die Generierung von SIBs für Business-APIs (wie in Abschnitt 3 beschrieben) im Mittelpunkt steht. Das Kapitel schließt mit einem konkreten Anwendungsbeispiel in Abschnitt 5.4.

5.1 Service-Integration mit SIBs

Die in Abschnitt 2.6.3 eingeführten SIBs bieten eine einfache und gute Möglichkeit der Umsetzung von domänen-spezifischen Business-Aktivitäten (siehe dazu Abschnitt 4.5). Durch ihre Vielseitigkeit ist es aber ebenso möglich, Skript-Aktivitäten (siehe Abschnitt 4.3) oder technische Service-Aktivitäten (siehe Abschnitt 4.4) mit ihnen umzusetzen.

Das jABC enthält eine kontinuierlich wachsende Sammlung häufig benutzter SIBs. Diese Sammlung trägt den Namen „Common-SIBs“ und ist taxonomisch organisiert. Hier existiert eine große Anzahl von SIBs aus sehr unterschiedlichen Anwendungsfeldern. Dadurch, dass viele SIBs bereits fertig zur Verfügung stehen und die Sammlung durch neue Projekte

immer weiter wächst, wird die Entwicklung neuer ausführbarer Prozesse zunehmend beschleunigt.

5.1.1 Skript-Aktivitäten

Eine Kategorie der „Common-SIBs“ heißt „Script-SIBs“. Wie der Name schon andeutet, entspricht dies einer Sammlung von Realisierungen der in Abschnitt 4.3 beschriebenen „Skript-Aktivitäten“. So gibt es beispielsweise ein SIB zur Ausführung von BeanShell-Skripten[1] und eines für Groovy-Skripte [2]. Außerdem existieren noch einige SIBs für verschiedene Template-Engines (Velocity[3], StringTemplate[4], FreeMarker[5]), welche auch als spezielle technische Service-Aktivitäten gesehen werden können, aber aufgrund der Tatsache, dass sie ein Skript (das Template) ausführen, in der Kategorie „Skript-Aktivitäten“ fallen. Die Template-Engine-SIBs kommen alle aus dem Genesys-Projekt (siehe Abschnitt 2.6.4.2).

5.1.2 Technische Service-Aktivitäten

Das jABC ist nicht nur ein Tool für Geschäftsprozesse im engeren Sinne ist, sondern umfasst ein erheblich weiteres Anwendungsspektrum. Wie in Abschnitt 2.6.4 beschrieben, handelt es sich um ein allgemeines Prozessframework, welches in den unterschiedlichsten Szenarien (Geschäftsprozesse, eingebettete Systeme, Code-Generierung, ...) Anwendung findet. Dies führt dazu, dass die Common-SIBs-Sammlung neben den oben erwähnten „Script-SIBs“ noch eine Reihe weiterer Kategorien enthält:

- Basic-SIBs (einfache Funktionalitäten wie z. B. ein einfaches if-SIB oder zur Platzierung von Datenobjekten im Ausführungskontext)
- IO-SIBs (Eingabe und Ausgabe über Bildschirm, Tastatur, Netzwerk, Dateisystem etc.)
- GUI-SIBs (Darstellung von Dialogen, Interaktion mit Benutzern)
- Collection-SIBs (Umgang mit Collection-Datentypen wie List, Map, etc.)
- Java-SIBs (sehr technische SIBs für Java-spezifische Aufgaben)
- MSOffice-SIBs (Manipulation von Word- und Excel-Dokumenten)
- OpenOffice-SIBs (Manipulation von OpenOffice-Dokumenten)

[1] http://www.beanshell.org/
[2] http://groovy.codehaus.org/
[3] http://velocity.apache.org/
[4] http://www.stringtemplate.org/
[5] http://freemarker.sourceforge.net/

- E-Mail-SIBs (Verschicken von E-Mails sowie Verwaltung eingehender E-Mails)

Die genannten SIBs können als technische Service-Aktivitäten aufgefasst werden. Sie besitzen insgesamt ein recht allgemeines Anwendungsfeld und sind somit besonders gut wiederverwendbar. Auf der anderen Seite sind sie oft zu kleinschrittig und technisch für einen Business-Experten.

5.1.3 Business-Aktivitäten

SIBs sind dafür gedacht, grobgranulare Arbeitseinheiten zu repräsentieren, die direkt und intuitiv von Business-Experten verstanden und eingesetzt werden können. Oft dienen sie sehr speziellen Zwecken (z. B. das Hinzufügen eines neuen Kunden zu einer CRM-Software), was sie zu idealen Kandidaten für die Umsetzung von domänenspezifischen Business-Aktivitäten macht. Durch ihre strukturierte Dokumentation, ihre illustrierenden Icons und ihre Validierungsregeln können Sammlungen von taxonomisch organisierten SIBs als eine graphische, domänenspezifische Business-Sprache verstanden werden. Da die Benutzung von SIBs als Business-Aktivitäten (besonders im Kontext der Integration von ERP-Systemen) die fortschrittlichste und geeignetste Variante der Service-Integration darstellt, befassen sich die folgenden Abschnitte im Wesentlichen mit diesem Konzept. In Abschnitt 5.2 werden Beispiele für vorgenommene manuelle Entwicklungen von Business-Aktivitäten als SIBs beschrieben, in Abschnitt 5.3.5 geht es dann um die Generierung von SIBs aus APIs mittels des Frameworks „InBuS“.

5.2 Manuelle SIB-Implementierungen

Der größte Teil der existierenden SIBs wurde manuell erstellt. Diese Herangehensweise ist besonders dann zu empfehlen, wenn die zu integrierende API von überschaubarer Größe ist. Auch bei extrem uneinheitlich gestalteten APIs kann dies nach wie vor der geeignetste Weg sein. Automatische oder halbautomatische Generierungen, wie sie in Abschnitt 5.3 beschrieben werden, spielen erst dann ihre Vorteile aus, wenn sich der Zusatzaufwand für die Entwicklung der Generatoren durch entsprechend große APIs auch auszahlt. Und selbst bei sehr großen APIs ist der erste Schritt zur Einarbeitung und praktischen Analyse der API meist die manuelle Implementierung kleinerer Beispiele. Diese Beispiele können dann sehr gut als Grundlage für die Automatisierung - z. B. mittels der in Abschnitt 5.3.1 beschriebenen Template-Engines - dienen.

5.2.1 Office-SIBs

Neben ERP-Systemen stellen Office-Lösungen wie MS Office oder Open-Office die wohl wichtigste Kategorie von Anwendungssoftware in Unternehmen dar. Bei Office-Lösungen handelt es sich dabei um Pakete, die typische Büro-Software wie eine Textverarbeitung, eine Tabellenkalkulation, eine Präsentationssoftware und oft noch weitere Anwendungen bündeln.

Besonders Tabellenkalkulationen wie Microsoft Excel bilden dabei oft den Kern einer Unternehmens-IT, und das nicht nur in kleinen Unternehmen. Der Vorteil einer solchen Lösung ist, dass auch Personen ohne besondere IT-Kenntnisse fähig sind, auf der Basis eigene Softwarelösungen zu erstellen. Werden jedoch auch zentrale Geschäftsprozesse „ad hoc“ mit einer Tabellenkalkulation umgesetzt, und zwar von Personen außerhalb einer IT-Abteilung, so spricht man auch von „Shadow-IT“ [Rad05]. Genau so wie die Schattenwirtschaft unerkannt von der Steuer existiert, gibt es auch eine Art „Schatten-IT“ im Unternehmen, die unerkannt von der IT-Abteilung und somit auch oft unerkannt für die Unternehmensleitung existiert. Geschäftsprozesse werden dann nicht wie es das Geschäftsprozessmanagement vorschreibt, graphisch modelliert, zentral vorgehalten und zum zentralen Element der IT-Umsetzung. Stattdessen werden „auf die Schnelle“ oft relativ prototypisch anmutende Lösungen entwickelt, die meist nicht dokumentiert und somit nur vom Ersteller selbst bedienbar sind. Dies hat eine Reihe von Problemen zur Folge. Da der Ersteller meist keine IT-Kenntnisse hat, sind die Lösungen oft sehr unfachmännisch erstellt und enthalten viele kleine „Programmiertricks“, die sich der Ersteller oft selbst ausgedacht hat. Somit sind sie extrem fehleranfällig. Da eine Tabellenkalkulation keine Prozesssteuerung enthält, ist nirgendwo hinterlegt, wie genau sie zu bedienen ist. Die Reihenfolge der typischen Schritte zur Erledigung einer Aufgabe ist meist nirgendwo dokumentiert. Das hat zur Folge, dass nur der Ersteller selbst sein Konstrukt bedienen kann und er sich somit selbst unersetzlich macht. Verlässt er das Unternehmen oder wechselt er nur die Abteilung, so kann dies katastrophale Folgen haben.

Um diese Probleme zu beheben, ist anzustreben, solche prototypischen Lösungen zu migrieren, und zwar hin zu prozessgesteuerten, serviceorientierten Systemen. Dabei können die existierenden Tabellenkalkulationen gut als Einstieg in die Anforderungsanalyse herangezogen werden. Da hier die Prozesse nicht enthalten sind, müssen ergänzend Interviews mit den Erstellern durchgeführt werden. Gibt es neben dem Ersteller noch weitere Benutzer, so können diese ebenfalls interviewt werden. Die Protokolle dieser Interviews können dann als Grundlage für die Prozessmodellierung dienen. Die Tabellenkalkulationen selbst dienen eher als

Anforderungsanalyse für die Datenseite.

Ist die Modellierung abgeschlossen, können die Prozesse technisch umgesetzt werden. Um einen „Big Bang“ zu vermeiden, kann an dieser Stelle eine schrittweise Migration vollzogen werden. Dazu wird zunächst die Tabellenkalkulation weiter als Daten-Backend genutzt und durch den automatisierten Geschäftsprozess gesteuert. Erst in einem späteren Schritt kann dann bei Bedarf die Tabellenkalkulation zum Beispiel durch ein Datenbanksystem oder gar ein ERP-System ersetzt werden. Es muss also möglich gemacht werden, ein Office-System service-orientiert in ein BPMS zu integrieren. Im besten Fall möchte man Business-Aktivitäten, die der Nutzung und Manipulation von Office-Dokumenten wie z. B. Tabellenkalkulationen dienen. Dies wurde in verschiedenen Projekten durchgeführt. Die Projektgruppe „Excelerate“ [ACD+12] befasste sich zum Beispiel genau mit diesen Thema, nämlich der Migration von Microsoft Excel-basierten Lösungen hin zu agilen service-orientierten Architekturen. Weiterhin wurden sowohl für Industrieprojekte als auch für interne Demonstrationszwecke weitere Integrationen von Office-Produkten durchgeführt. Textverarbeitungssoftware wurde in Prozessen oft zur einfachen Generierung von Berichten verwendet. Im Folgenden wird auf die unterschiedlichen integrierten Systeme und die verschiedenen Herangehensweisen der Integration eingegangen. In Abschnitt 5.2.1.1 geht es um die Integration von MS-Office auf zwei unterschiedliche Arten, in Abschnitt 5.2.1.2 folgt dann eine Erläuterung der Integration von OpenOffice bzw. LibreOffice.

5.2.1.1 Microsoft Office

Die Ansteuerung von Microsoft Office von außen ist prinzipiell möglich, jedoch nicht immer ganz unproblematisch. Es gibt grundsätzlich dabei zwei verschiedene Ansätze. Zum einen kann man über die COM-basierte API auf Office zugreifen, zum anderen ist es möglich, die Office-Dateien zu manipulieren, ohne dass die Office-Software selbst mit einbezogen wird.

Ansteuerung über COM Prinzipiell existiert eine API, welche den gesamten Funktionsumfang abdeckt. Dies ist allein schon deshalb wichtig, da es mit VBA[6] eine integrierte Skriptsprache gibt, mit der Makros verfasst werden können. Mit der Hilfe dieser Makros lassen sich alle Vorgänge innerhalb von Microsoft Office automatisieren.

Technisch setzt diese API dabei auf Microsoft's Technologie COM[7] bzw.

[6]VBA: Visual Basic for Applications
[7]COM: Component Object Model

dessen verteilte Variante DCOM[8] auf. Bei letzterem handelt es sich um eine proprietäre, objektorientierte RPC-Lösung.

Aus mehreren Gründen ist Benutzung der COM-Schnittstelle problematisch. Da es eine proprietäre Microsoft-Lösung ist, wird die Technologie nicht direkt in Microsoft-fremden Umgebungen wie Java unterstützt. Möchte man von Java aus auf eine COM-API zugreifen, so müssen so genannte Java-COM-Bridges zum Einsatz kommen. Hier existieren zum Beispiel die Implementierungen JaCoB[9], J-Interop[10] oder Jawin[11]. Eine solche Adaptersoftware kann die Lücke jedoch nie ganz schließen. Möchte man zum Beispiel ein Office-System ansteuern, welches auf dem lokalen System läuft, so müssen die Anmeldedaten des entsprechenden Benutzers (also Benutzername und Passwort) meist angegeben werden, obwohl der aufrufende Java-Prozess schon als Windows-Benutzer angemeldet ist.

Ein weiterer Nachteil dieses Verfahrens ist (im Unterschied zum Dokumentenverfahren in folgendem Abschnitt), dass eine Instanz der Office-Software gestartet und gesteuert werden muss. Man muss also zum einen ein Windows-System mit installiertem Office-Paket zur Verfügung haben, zum anderen muss die Software für jede Benutzung gestartet werden. Dieser Vorgang kann manchmal recht langsam sein, da es sich bei der Office-Suite um ein recht umfangreiches Softwarepaket handelt.

Weiterhin ist es ganz besonders zu erwähnen, dass die Office-API sich von einer Office-Version zur nächsten erheblich ändern kann. So ist man nie sicher, dass ein API-Aufruf nach einem Versionswechsel immer noch funktioniert.

Vorteilhaft an dieser Art der Ansteuerung ist die Verfügbarkeit einer vollständigen API, die die Gesamtheit aller Funktionen der Office-Suite abdeckt. Auch gibt es einen recht einfachen Weg, herauszufinden, welche konkreten Aufrufe nötig sind, um ein bestimmtes Verhalten zu bewirken. Dazu kann sehr gut die Makro-Rekorder-Funktion verwendet werden. Dabei wird einfach ein Aufnahme-Vorgang in der GUI von Office gestartet, man klickt sich durch den gesamten Vorgang und stoppt anschließend die Aufnahme. Im Hintergrund wurde nun während der Aufnahme VBA-Code erzeugt, welcher genau diesen Vorgang durchführt. Man hat also eine sehr intuitive und einfache Methode der Service-Discovery.

Nutzung von Apache POI Eine gute Alternative zur Nutzung der COM-Schnittstelle ist die Manipulation der Office-Dokumente ohne die Nutzung der Office-Suite selbst. Dazu kann zum Beispiel die Software

[8]DCOM: Distributed Component Object Model
[9]http://sourceforge.net/projects/jacob-project/
[10]http://j-interop.org/
[11]http://jawinproject.sourceforge.net/

des Projektes „POI“[12] von der Apache Software Foundation[13] eingesetzt werden. Dabei handelt es sich um eine Java-Bibliothek zum Lesen und Manipulieren von Office-Dokumenten. Unterstützt werden hier die Dateiformate für Word, Excel, Powerpoint, Outlook, Visio und Publisher. Als besonders ausgereift gilt die Implementierung des Excel-Formats. Insgesamt ist hier zu berücksichtigen, dass die Lösung nicht von Microsoft selbst stammt, sondern das Ergebnis von Reverse-Engineering ist. Es werden nie alle Funktionen und Eigenschaften der Dateiformate abgedeckt und bei jeder neuen Office-Version dauert es naturgemäß eine Weile, bis die Implementierer es auch für diese Version wieder geschafft haben, das Dateiformat zu „entschlüsseln“.

Unter Nutzung von Apache POI wurde unter meiner Anleitung eine Sammlung von SIBs zum Lesen und Bearbeiten von Excel-Dokumenten erstellt. Folgende SIBs wurden dabei entwickelt:

- CreateXLSDocument
- OpenXLSDocument
- SaveXLSDocument
- ReadXLSDocumentCell
- EditXLSDocumentCell
- FormatXLSDocumentCell

Es existieren also SIBs zum erstellen, öffnen und speichern von Dokumenten und SIBs zum lesen, bearbeiten[14] und formatieren[15] von Zellen. Diese SIBs sind demnach sehr kleinschrittig und technisch, gehören somit in die in Abschnitt 4.4 beschriebene Kategorie der technischen Service-Aktivitäten. Solche SIBs werden meist nicht vom Anwendungsexperten selbst genutzt und verwendet, sondern müssen zunächst in einem technischen Modell zu einem konkreten Anwendungsfall kombiniert werden. Dieses Modell kann dann wieder als Ganzes bereitgestellt und per Makro-SIB in ein übergeordnetes Modell eingebettet werden.

5.2.1.2 OpenOffice / LibreOffice

Bei der Ansteuerung der freien Office-Lösung „OpenOffice“[16] (oder dessen Fork „LibreOffice“[17], findet man eine gänzlich andere Situation vor

[12]POI: (ursprünglich) Poor Object Implementation
[13]http://poi.apache.org/
[14]Zell-Typen: Numeric, String, Formula, Blank, Boolean, Error
[15]Schriftgröße, unterstrichen, fett, kursiv
[16]http://www.openoffice.org/de/
[17]http://de.libreoffice.org/

als beim oben beschriebenen Microsoft Office. Hier gibt es jeweils eine offizielle Java-API[18]. Es ist also von vornherein vorgesehen, diese Systeme plattformunabhängig ansteuern zu können. Dies erleichtert eine serviceorientierte Einbindung in einen Geschäftsprozess enorm. Da es sich hier um eine API handelt, ist der Ansatz prinzipiell eher mit dem COM-Ansatz bei Microsoft Office zu vergleichen, nicht mit dem POI-Ansatz.

Es existiert am Lehrstuhl 5 eine Implementierung von OpenOffice-SIBs, die von Martin Sugioarto implementiert wurden. Diese SIB-Sammlung enthält SIBs in drei verschiedenen Unterkategorien:

- Allgemeine SIBs (Kategorie „Common"):
  - CloseDocument: Schließt das Dokument
  - NewDocument: Erzeugt ein neues Dokument
  - PrintDocument: Druckt ein Dokument
  - SaveDocuement: Speichert ein Dokument
  - SaveDocumentAsPDF: Exportiert das Dokument als PDF
  - StartOffice: Startet eine Instanz von OpenOffice
- Tabellenkalkulation Calc (Kategorie „Calc"):
  - CalculateColumnSum: Berechnet die Summe einer Spalte
  - GetCellValue: Berechnet den Wert einer Zelle
  - InsertNewSheet: Fügt eine neue Tabelle ein
  - SetCellValue: Setzt den Wert (oder die Formel) einer Zelle
  - SetCurrentSheet: Wählt die aktuelle Tabelle aus
- Textverarbeitung Writer (Kategorie „Writer"):
  - InsertTable: Fügt eine Tabelle ein
  - MoveCursor: Bewegt den Cursor an eine vorgegebene Stelle
  - ReplaceText: Sucht nach einem vorgegebenen Text und ersetzt alle Vorkommen durch einen gegebenen neuen Text
  - SetCursorProperties: Setzt die aktuellen Eigenschaften des Cursors (Schriftart und -größe, Farbe, Unterstrichen, ...)
  - WriteText: Fügt Text an der aktuellen Cursor-Position ein

Die OpenOffice-SIBs wurden in zahlreichen Demos und Beispielprozessen eingesetzt. Die Writer-SIBs können zum Beispiel sehr gut als einfache Möglichkeit der Report-Generierung benutzt werden. Dazu können Vorlagen als OpenOffice-Dokumente erstellt werden, wobei an bestimmten

[18] http://www.openoffice.org/api/ bzw. http://api.libreoffice.org/

Stellen entsprechend gekennzeichnete Platzhalter eingefügt werden. Diese Platzhalter sollten eindeutig gekennzeichnet sein, zum Beispiel durch eine Einfassung in jeweils zwei aufeinander folgende Dollarzeichen wie in folgendem Beispiel:

```
Die Gesamtpreis beträgt $$GESAMTPREIS$$ Euro.
```

Nun kann durch die Benutzung des SIBs „ReplaceText" dieses Dokument mit Daten gefüllt werden. Die Erzeugung des Endresultats ist demnach vergleichbar mit einer sehr einfachen Variante der Template-basierten Generierung (siehe dazu Abschnitt 5.3.1).

5.2.2 SAP-SIBs

Wie in Abschnitt 2.5 beschrieben, bilden ERP-Systeme oft das Rückgrat einer Unternehmens-IT-Infrastruktur. Daher ist es von besonderer Wichtigkeit, diese Systeme in den Geschäftsprozessen zu berücksichtigen und die von den ERP-Systemen bereitgestellten Services in die Automatisierungen der Geschäftsprozesse zu integrieren. SAP-ERP ist dabei als weltweit führende Lösung ein besonders interessantes Studienobjekt. In Abschnitt 3.2.1 wurde die BAPI-Technologie vorgestellt und untersucht, wie damit auf das SAP-System zugegriffen werden kann. In Abschnitt 3.2.2 folgte eine Untersuchung der Web-Service-basierten eSOA-Lösung von SAP.

Die erste Sammlung von SIBs, welche SAP-ERP-Services aufrufen, wurde manuell implementiert. Als technische Grundlage wurde dabei die SAP-BAPI und der Java-Connector (JCo) gewählt.

Wenn man JCo[19] direkt benutzt, damit also manuell in Java einen BAPI-Aufruf implementiert, sieht der dabei entstehende Code zum Beispiel wie hier in Listing 5.1 dargestellt aus (Beispiel aus [MBD+11]):

Listing 5.1: Java-Code zum Aufruf einer BAPI-Methode mit JCo

```
1 ...
2 JCO.Client client = JCO.createClient("001","user",
3     "apassword","EN","some.hostname.com","00");
4 client.connect();
5 JCO.Repository repo = new JCO.Repository(
6     "myRepository",client);
7 IFunctionTemplate functionTlt =
8 repo.getFunctionTemplate("BAPI_MATERIAL_SAVEDATA");
9 JCO.Function function = functionTlt.getFunction();
```

[19] Hier wurde JCO in der Version 2.0 verwendet.

```
10  ParameterList importParameters =
11      function.getImportParameterList();
12  JCO.Structure headData =
13      importParameters.getStructure("HEADDATA");
14  headData.setValue("M1234","MATERIAL");
15  headData.setValue("HAWA","MATL_TYPE");
16  headData.setValue("1","IND_SECTOR");
17  JCO.Structure clientData =
18      importParameters.getStructure("CLIENTDATA");
19  clientData.setValue("PCE","BASE_UOM_ISO");
20  clientData.setValue("002","MATL_GROUP");
21  JCO.Structure clientDataX =
22      funcion.getImportStructure("CLIENTDATAX");
23  clientDataX.setValue("X","BASE_UOM_ISO");
24  clientDataX.setValue("X","MATL_GROUP");
25  JCO.Parameter tableParameters =
26      function.getTableParameterList();
27  JCO.Table descrTable =
28      tableParameters.getTable("MATERIALDESCRIPTION");
29  descrTable.appendRow();
30  descrTable.setValue("e","LANGU");
31  descrTable.setValue("computer mouse", "MATL_DESC");
32  client.execute(function);
33  JCO.ParameterList exportParameters =
34      function.getExportParameterList();
35  JCO.Structure returnStructure =
36      exportParameters.getStructure("RETURN");
37  if (!(returnStructure.getString("TYPE").equals("") ||
38      returnStructure.getString("TYPE").equals("S"))) {
39          String errorMessage =
40              returnStructure.getString("MESSAGE");
41          throw new Exception(errorMessage);
42      }
43  System.out.println("Material successfully added!");
44  client.disconnect();
45  ...
```

An diesem Beispiel kann man sehr gut sehen, warum es nicht erstrebenswert ist, für jeden Service-Aufruf eine entsprechende manuelle Implementierung direkt mit JCo durchzuführen. Der Code ist insgesamt sehr lang, technisch, kompliziert und an vielen Stellen Java-untypisch.

Obwohl es sich bei JCo um eine Java-Bibliothek handelt, wurden hier viele Dinge auf eine Weise umgesetzt, die nicht den Sprachkonventionen und üblichen Gepflogenheiten unter Java-Programmierern entsprechen. Das führt unweigerlich zu einer Verletzung des ITSy-Prinzips (siehe Abschnitt 2.7), da es grundsätzlich nicht einfach ist, entgegen seiner Ge-

wohnheiten und Erfahrungen zu arbeiten. Was zunächst auffällt (z. B. in Zeile 2 des Beispiels) ist, dass fast die gesamte Klassenbibliothek innerhalb einer Klasse (die Klasse `JCO`) organisiert ist. Die einzelnen Klassen sind also jeweils Unterklassen dieser Hauptklasse. In Zeile 7 sieht man, dass ein Interface den Namen `IFunctionTemplate` trägt. Das Präfix `I` steht hier für „Interface“. Diese Art der Namensgebung wird auch als „ungarische Notation“ bezeichnet und ist im Java-Umfeld eher unüblich. Geläufiger ist diese Notation z. B. im Windows-Umfeld, also z. B. in der .NET-Klassenbibliothek[20]. In den Zeilen 14-16, 19-20 und 23-24 werden jeweils Schlüsselworten Werte zugewiesen, es werden also Key-Value-Paare definiert. Wie die Bezeichnung „Key-Value“ schon suggeriert, ist hier die Reihenfolge Key-Value üblich, JCo nutzt jedoch grundsätzlich die Reihenfolge Value-Key. Der eigentliche Funktionsaufruf findet in Zeile 32 statt. Hier wird die Methode `execute()` aufgerufen, welche weder Werte übergeben bekommt noch Werte zurückgibt. Der Umgang mit Daten wird vorher bzw. nachher durch entsprechende (mit `get` beginnende) Methoden[21] explizit durchgeführt. Im Java-Umfeld (z. B. bei durch JAX-WS generierte SOAP-Web-Service-Stubs) ist es eher üblich, der aufrufenden Methode die Werte zu übergeben und den Rückgabewert direkt durch diese Methode zu erhalten.

Manche technischen Eigenschaften aus dem SAP-Technologie-Umfeld (also von BAPI, RFC und ABAP) werden in der JCo-API nicht versteckt, sondern wie sie sind angeboten. Diese Eigenschaften sind ebenfalls nicht Java-typisch. Am wichtigsten ist hierbei wohl die Beschränkung der komplexen Datenstrukturen auf Structure und Table. Dies ist weit von einer Objektorientierung entfernt. Ein weiterer Punkt ist, dass es manchmal neben den eigentlichen Datenstrukturen noch zusätzlich sogenannte „X-Leisten“ gibt. Dabei handelt es sich um eine Checkliste (also eine Liste von ankreuzbaren Feldern), die für jedes Feld einer Datenstruktur angibt, ob die hier ein neuer Wert angegeben wird, der berücksichtigt werden soll. Im Beispiel ist die Behandlung einer solchen X-Leiste in den Zeilen 21 bis 24 zu sehen. Hier existiert zur Datenstruktur `CLIENTDATA` noch die X-Leiste `CLIENTDATAX`. Für alle Felder dieser Datenstruktur muss nun der Wert auf `"X"` gesetzt werden. Auch ist hier zu sehen, wie intern der fehlende Boolean-Datentyp abgebildet wird, und zwar mit einer Zeichenkette, die genau ein Zeichen lang ist und entweder ein X für `true` oder ein Leerzeichen für `false` enthält.

Bei der Behandlung von Fehlern kommt ebenfalls das SAP-typische zum Vorschein und ersetzt die Java-typische Herangehensweise. Kommt es beim Aufruf der Funktion zu einem Fehler, so wird nicht Java-typisch eine Exception geworfen. Der Funktionsaufruf wird stattdessen ganz nor-

[20] http://msdn.microsoft.com/de-de/library/aa260976(VS.60).aspx

[21] z. B. `getImportParameterList()`, `getExportParameterList()`, `getTableParameterList()`

mal abgearbeitet. Nach dem Funktionsaufruf muss nun die Structure `RETURN` analysiert werden. Hier wird wiederum mit entsprechenden Codes gearbeitet, die man der SAP-Dokumentation entnehmen muss.

Neben der Missachtung von Java-Standards gibt es noch ganz allgemeine Dinge, die zu unnötig komplexem Code führen. So wird die Bedeutung des Repository-Objektes in Zeile 5 nicht wirklich klar. Hier muss ein Repository angelegt werden, welches mit einem Namen versehen werden muss. Dieser Name ist jedoch beliebig. Er hat keine Auswirkung auf die weitere Ausführung. Es gibt sicherlich einen technischen Grund für dieses Objekt, welcher jedoch offensichtlich nicht in der von SAP verfassten Dokumentation hinreichend erläutert wird. Für viele Standardfälle - so z. B. für alle Funktionsaufrufe, die im Rahmen dieser Arbeit getätigt wurde - ist diese Angabe überflüssig.

Wie schon in Abschnitt 3.1.2 beschrieben, sind intuitive Benennungen ein entscheidender Aspekt bei dem Design einer gut benutzbaren API. Bei der JCo-API sind hier vor allem die Namen `IFunctionTemplate` und `Function` zu kritisieren. Ein `IFunctionTemplate` ist das Objekt, welches die Funktion repräsentiert, man hätte es also auch einfach `Function` nennen können. Das, was bei JCo `Function` heißt, ist das Objekt, welches die konkreten Eingabe- und Ausgabedaten eines Funktionsaufrufs enthält. Eine passendere Benennung wäre hier also `FunctionCall` gewesen.

All die genannten Probleme und Optimierungsmöglichkeiten im Umgang mit JCo führten dazu, dass im Rahmen dieser Arbeit eine neue, zusätzliche Bibliothek als Wrapper um JCo entwickelt wurde, die die technischen Eigenheiten von JCo versteckt. Der erste Schritt bestand dabei daraus, aus JCo eine Java-typische Library machen. Dabei wurde natürlich auch auf grundsätzliche Anforderungen an gutes API-Design gedacht. Einfache Dinge sollten einfach und mit wenig Code beschreibbar sein.

Auf der Basis dieser technischen Bibliothek konnte anschließend eine Sammlung fachlicher Java-Services entwickelt werden. Diese stellen also das technische Backend für die in Abschnitt 4.5 beschriebenen Business-Aktivitäten dar. Anstatt also bei der Benutzung der Bibliothek mit Funktionen, Parametern und Werten zu arbeiten, stehen nun stattdessen Objekte wie Kunden, Materialien oder Mitarbeiter im Mittelpunkt.

Beide genannten Schichten (zum einen die Java-typischen Wrapper, zum anderen Business-Funktionen) werden in der so genannten „SAP-Lib" zusammengefasst. Mit dieser Java-Bibliothek kann also sehr einfach von Java aus auf ein SAP-ERP zugegriffen werden, und zwar weitesgehend ohne tiefgehende SAP-Kenntnisse. Diese Bibliothek ist komplett unabhängig vom jABC und kann somit in beliebigen Java-Projekten genutzt werden.

Das oben aufgeführte Beispiel, in dem ein neues Material hinzugefügt

wurde, sieht also bei der Benutzung der SAP-Lib wie in Listing 5.2 aus:

Listing 5.2: BAPI-Aufruf mit SAP-Lib

```
...
SAPConnection connection =
    new SAPConnection(jcoProperties);
Material material = new Material();
material.setMaterialNumber("M1234");
material.setMaterialType("HAWA");
material.setIndustrySector("1");
material.setUnitOfMeasure("PCE");
material.setMaterialGroup("002");
material.setDescription("computer mouse");
try {
    MaterialBAPI.addMaterial(connection, material);
} catch (SAPException e) {
    e.printStackTrace();
}
...
```

Hier sieht man, dass der Code zum einen deutlich kürzer ist als in Listing 5.1, zum anderen ist er deutlich intuitiver und Java-typischer. Das Material-Objekt ist nun ein Objekt im objektorientierten Sinne, der eigentliche Aufruf befindet sich in einer Zeile und bei einem auftretenden Laufzeitfehler wird eine entsprechende Exception geworfen.

Zusammen mit der SAP-Lib existieren noch weitere Pakete, durch die diese Funktionen in das jABC integriert werden:

- **SAP-Forms**: Dieses Paket enthält eine Sammlung von Dialogen, also Eingabemasken und Anzeigedialoge, die mit Java-Swing erstellt wurden. Diese Dialoge wurden spezifisch für bestimmte Funktionen entworfen. Oft ist es so, dass in einem Prozess vor einem Funktionsaufruf Daten vom Benutzer erwartet werden und nach einem Funktionsaufruf entsprechende Ergebnisse am Bildschirm präsentiert werden sollen. Genau für diese Zwecke werden diese Dialoge eingesetzt.
- **SAP-SIBs**: Dieses Paket enthält SIBs, welche die Services der SAP-Lib aufrufen oder Dialoge der SAP-Forms anzeigen. Es handelt sich bei diesen SIBs also um domänenspezifische Business-Aktivitäten (siehe Abschnitt 4.5).
- **SAP-Demo**: Dieses Paket enthält Beispielprozesse, die zeigen, wie die SAP-SIBs verwendet werden können, um einen ausführbaren Geschäftsprozess darzustellen.

5.2.3 Weitere SIB-Implementierungen

Neben den in Abschnitt 5.2.1 beschriebenen Office-SIBs und den in Abschnitt 5.2.2 beschriebenen SAP-SIBs existieren noch viele weitere SIB-Sammlungen, die manuell erstellt wurden und für die Modellierung von Business-Prozessen benutzt werden können. Im folgenden werden stellvertretend drei verschiedene Sammlungen kurz vorgestellt.

Common SIBs: Die Sammlung der „Common SIBs“ wurde bereits in Abschnitt 5.1 erwähnt. Diese SIBs sind größtenteils eher technischer Natur und werden bei der Entwicklung von Geschäftsprozessen eher auf sehr niedrigen Ebenen der Prozesshierarchie, also in technischen Prozessen verwendet. Viele dieser SIBs sind tendenziell eher kleinschrittig und erfordern zum Teil die Kenntnis bestimmter Technologien (wie z. B. der Expression-Language).

Mail-SIBs: Eine große Rolle im Technologieumfeld von Unternehmen spielt die E-Mail. Zu diesem Thema wurde eine Diplomarbeit angefertigt [Saß07]. Dennis Saßmannshausen entwickelte dafür eine Sammlung von entsprechenden „Mail-SIBs“, zum einen zum Versenden von E-Mails, zum anderen zur Verwaltung eingehender E-Mails. Zur Implementierung dieser SIBs wurde die Java-Bibliothek JavaMail[22] eingesetzt, welche Teil der Java Enterprise Edition ist.

Integration von SAP eSOA und Google Apps: Im Rahmen der Vorlesung „Aktuelle Themen der Dienstleistungsinformatik“ wurden in den Wintersemestern 2010/2011, 2011/2012 sowie 2012/2013 in Kleingruppen bestehend aus jeweils zwei bis vier Studenten praktische Projekte mit dem jABC umgesetzt, in denen auch SIBs entwickelt und damit Prozesse erstellt wurden. Zwei dieser Projekte sind dabei einem identischen, fiktiven Szenario zugeordnet. Dieses ist wie folgt beschrieben:

> „Die Firma IDES AG verwaltet alle Daten des Unternehmens mittels SAP-Software, konkret mit einem SAP-ERP. Dieses System hat sich auf der einen Seite als sehr komplex in der Bedienung erwiesen, auf der anderen Seite bedeutet der Einsatz von SAP-ERP einen erheblichen Kostenaufwand. Um in Zukunft Kosten zu senken, sich weniger abhängig von SAP zu machen und eine einfacher zu bedienende Software zu haben, plant die IDES AG nun eine schrittweise Migration auf kostenlose Cloud-Dienste der Firma Google. So können z. B.

[22]http://www.oracle.com/technetwork/java/javamail/index.html

> Personendaten in Google Contacts, Aufgaben in Google Tasks, Termine in Google Calendar und Dokumente in Google Docs (bzw. Google Drive) verwaltet und bearbeitet werden."

In dem einen Projekt ging es um einen Prozess, in dem Kundendaten sukzessive von SAP nach Google Contacts migriert werden, und zwar immer nur nach Bedarf, damit keine veralteten Daten blind kopiert werden. Dieser Prozess sollte danach noch in zwei weiteren Varianten entwickelt werden, und zwar für Lieferanten sowie für Mitarbeiter.

Im zweiten Projekt ging es um Kundenaufträge, die aus SAP herausgelesen und in Tabellenkalkulationen bei Google Spreadsheets transformiert werden sollten.

In beiden Projekten mussten also sowohl SIBs für die entsprechenden Google Apps als auch SIBs für SAP entwickelt werden. Google bietet für seine Apps die so genannten „Google Apps Application APIs"[23]. Dies ist eine Sammlung von APIs für alle von Google angebotenen Webapplikationen. Die API ist REST-basiert und bietet zusätzlich Client-Bibliotheken (mit Namen „gdata") für verschiedene Programmiersprachen (Java, JavaScript, .NET, PHP, Python, Objective-C) an[24]. In diesem Fall wurde die Java-Bibliothek eingesetzt. Diese Client-Bibliothek deckt in einer gut zu benutzenden Java-API das gesamte Leistungsangebot der Google Apps ab. Die Bereitstellung einer solchen Bibliotheken-Sammlung in gleich sechs Programmiersprachen stellt einen erheblichen Aufwand für den Anbieter Google dar. Bei jeder Änderung der REST-API müssen auch alle Client-Bibliotheken angepasst werden. Auf der anderen Seite ist die Situation für die Benutzer der API so besonders komfortabel. Man muss sich keine Gedanken über entfernte Funktionsaufrufe, Sicherheitsaspekte oder Netzwerkthemen im Allgemeinen machen. Stattdessen kann einfach die lokale Java-API genutzt werden. Der Entwicklungsaufwand liegt hier also beim Anbieter, nicht beim Benutzer. Dies ist Google möglich, da die angebotenen Dienste ein relativ festes Angebot an Services besitzt. Bei ERP-Systemen wie SAP-ERP oder CRM-Systemen wie Salesforce wäre dieser Ansatz kaum praktikabel. Zum einen sind diese APIs erheblich größer, zum anderen treten hier deutlich häufiger Änderungen und Anpassungen (z. B. im Customizing) auf. Salesforce bietet daher zwar auch eine eigene Client-Bibliothek an, diese generiert jedoch immer erst den Client-seitigen Code aus der API-Beschreibung.

Bei SAP wurde dabei die SOAP-basierte eSOA-Technologie eingesetzt (siehe dazu auch Abschnitt 3.2.2). Bei SAP-eSOA wird zu den angebotenen SOAP-Webservices keine Client-Library angeboten. Stattdessen

[23]https://developers.google.com/google-apps/app-apis
[24]https://developers.google.com/gdata/docs/client-libraries

kann man sich mit Standard Webservice-Frameworks wie JAX-WS oder Apache Axis Client-Stubs generieren. Bei der Umsetzung der Projekte kamen die Studenten relativ schnell zu der Erkenntnis, dass es sich bei eSOA um eine sehr große und auch sehr inkonsistente API handelt. Damit wurden die Ergebnisse aus Abschnitt 3.2.2 bestätigt. Es wurde eigentlich erwartet, dass nach der Umsetzung des Kundenprozesses eine analoge Entwicklung für Lieferanten und Mitarbeiter sehr einfach durch die Ersetzung entsprechender Services möglich sei. Es stellte sich heraus, dass dies nicht möglich ist, da Kunden, Lieferanten und Mitarbeiter in SAP-ERP grundsätzlich sehr verschieden behandelt werden. Auch die eSOA-API bot hier sehr verschiedene Services an. Folgende Services wurden angeboten:

- Lieferant
  - Find Supplier by Name and Address
  - Read Supplier Basic Data
- Mitarbeiter
  - Find Employee by Elements
  - Find Employee Address by Employee
- Kunde
  - Find Customer by Elements

Nicht nur die Benennung der Services ist jeweils sehr unterschiedlich. In allen Fällen ist es gewünscht, nach einer Suche Listen von Personen oder Organisationen zu erhalten. Welche Informationen jedoch pro Person oder Organisation mitgeliefert wird, ist sehr unterschiedlich. So wird die Adresse nur manchmal direkt mit angegeben. Fehlt sie, ist es oft nötig, für die entsprechende ID separat noch einmal die Adressinformation zu erfragen. Möchte man also eine Liste von z. B. 100 Adressen erhalten, so ist im einen Fall nur ein Aufruf nötig, im anderen 101 Aufrufe. Da letzteres zu einer sehr trägen Benutzeroberfläche mit nicht akzeptablen Wartezeiten führen würde, haben sich die Studenten dazu entschieden, hier die graphische Benutzeroberfläche entsprechend anzupassen. Hier werden zunächst lediglich die Namen der Personen angezeigt und erst bei einer konkreten Auswahl per Mausklick wird die entsprechende Adresse angefragt und angezeigt. Es zeigt sich also, dass das Design einer API auch gravierende Auswirkungen bis zum Endbenutzer haben kann.

5.3 SIB-Generierung

Liegt eine große Sammlung relativ gleichförmig strukturierter Services vor, wie es zum Beispiel bei den APIs der ERP-Systeme der Fall ist, so

ist es sinnvoll, die Integration nicht gänzlich manuell vorzunehmen, sondern sich durch Generierungstechniken unterstützen zu lassen. Die Automatisierung der Service-Integration führt so zum einen dazu, dass ein einzelner Service mit viel weniger Aufwand, also viel schneller vorgenommen werden kann. Dies führt somit direkt zu schnellen Reaktionszeiten der IT, Prozesse können also sehr agil an sich ändernde Bedürfnisse angepasst werden. Zum anderen ist die Integration eines einzelnen Services von technischen Details befreit, wodurch es auch Nicht-IT-Experten ermöglicht wird, die Integration vorzunehmen.

Im Kontext von XMDD und jABC bedeutet eine Automatisierung der Service-Integration eine automatische oder halbautomatische Generierung von SIBs. Dazu gibt es bereits seit langem unterschiedliche Ansätze, die jeweils in unterschiedlichen Szenarien ihre Daseinsberechtigung besitzen.

SIB-Creator-Plugin: Das jABC-Plugin „SIB-Creator“ dient zur automatischen Generierung von SIBs. Dabei werden hier zwei ganz unterschiedliche Bedienkonzepte und damit verbundene Vorgehensweisen angeboten:

1. SIB-Creator-GUI
2. SIB-Creator-API

Die graphische Benutzeroberfläche des Plugins ermöglicht die Definition von SIBs direkt in einem entsprechenden Dialog im jABC. In Abbildung 5.1 ist ein entsprechender Screenshot zu sehen. Im oberen Bereich gibt man den gewünschten Java-Package-Namen sowie den Klassennamen an. Die Schaltfläche rechts daneben dient zur Auswahl eines Icons. Darunter befinden sich Bereiche zur Angabe der Parameter und Branches. Für alle angegebenen Elemente kann über das Kontextmenü ein Text-Editor-Fenster zum Verfassen der Dokumentation geöffnet werden. Über die Schaltfläche „Interfaces“ können die zu implementierenden Interfaces[25] angegeben werden. Das Ergebnis der Generierung ist kein ausführbares SIB, sondern lediglich eine Art „Stub“, welcher noch mit einer Implementierung versehen werden muss.

Zusätzlich existiert auch noch ein Dialog zur Auswahl einer Java-Methode, zur welcher ein entsprechender Service-Adapter generiert wird. Somit kann also auf der einen Seite ein SIB-Stub, auf der anderen Seite der Service-Adapter generiert werden. Um beides zusammen zu bringen und das SIB voll funktionsfähig zu machen, ist zum Schluss jedoch immer noch manuelle Arbeit nötig. Dies wird nach wie vor traditionell im Sourcecode des SIBs gemacht.

[25]Zum Beispiel für den LocalChecker, einen Genesys-Code-Generator oder den Tracer

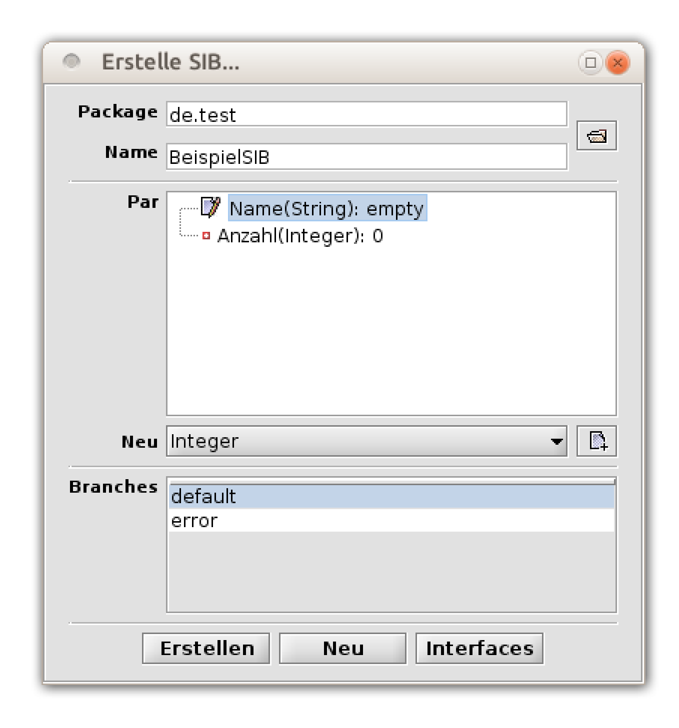

Abbildung 5.1: SIB-Creator-GUI

Die SIB-Creator-API ist nicht einfach eine API zu den Funktionen die auch per GUI angeboten werden. Stattdessen wird hier das Ziel der vollautomatischen Generierung ausführbarer SIBs verfolgt. Hiermit können zu existierenden Java-Methoden ausführbare SIBs erzeugt werden. Alle nötigen Informationen und Metadaten müssen dabei per API angegeben werden.

Um ein SIB zu generieren, müssen zunächst entsprechende Datenobjekte erzeugt werden, die alle nötigen SIB-Informationen enthalten. In einem `ClassWrapper`, einem `MethodWrapper` und potentiell mehreren Instanzen der Klasse `ParameterWrapper` werden alle nötigen SIB-Informationen angegeben. Dazu gehört neben der Auswahl der Java-Methode zum Beispiel auch der Klassenname des neuen SIBs, die UID, das Icon und Dokumentationstext. Die Benutzung der SIB-Creator-API ist naturgemäß deutlich technischer und komplexer als die der GUI. Erst nach einiger Einarbeitungszeit kennt man die Aufgaben der diversen Wrapper-Klassen und Factory-Klassen. Der Benutzer sollte sich außerdem mit Java-Reflection auskennen. Zur Ausführung der Generierung wird die Klasse `HWTQueueExecutor` eingesetzt, deren Benutzung ebenfalls beherrscht werden muss.

Aufgrund dieser recht komplexen Benutzung wurde in dieser Arbeit auf die Benutzung der SIB-Creator-API verzichtet. Stattdessen wurde ein Ansatz gewählt, der rein auf den Einsatz von Templates der Template-

Engine „StringTemplate“ setzt. Darauf wird in Abschnitt 5.3.1 genauer eingegangen. Mit der SIB-Creator-API wäre außerdem eine Übertragung des Konzeptes vom jABC auf andere Prozessframeworks nicht oder nur schwer möglich gewesen. Diese wird in Kapitel 7 beschrieben.

Generierung von R-SIBs: Ein Beispiel für die Anwendung der SIB-Creator-API ist die Diplomarbeit von Clemens von Musil [Mus09]. Hier geht es um die Integration der Statistiksprache R[26] in das jABC. Dazu müssen direkt ausführbare SIBs generiert werden, welche wiederum R-Code aufrufen. Diese SIBs sollen dabei direkt durch den Anwendungsexperten[27] benutzbar sein. Die Generierung der SIBs erfolgt hier nicht „auf Vorrat“ sondern immer nur bei Bedarf. Zur Anbindung von R aus Java heraus existiert bereits das Java R Interface[28] sowie die Technologie Rserve[29]. Rserve besteht aus einem TCP/IP-Server sowie Client-Bibliotheken für R, C++ und Java. Diese Technologie (mit dem Java-Client) wurde für die R-SIBs eingesetzt. Ein R-SIB muss nun R-Anweisungen erzeugen, diese über Rserve ausführen lassen und das Ergebnis der Ausführung in den Kontext legen. Die Service-Adapter wurden mit der Template-Engine Apache Velocity generiert, für die SIBs selbst wurde die SIB-Creator-API genutzt. In der Praxis stellte sich später heraus, dass die generierten SIBs nach wie vor recht technisch sind, was dazu führte, dass diese Generierungstechnologie nicht sehr häufig zum Einsatz kam.

SIB-Generierung in jETI: In jETI[30], werden mit dem Tool WSDL2SIB vollautomatisch direkt ausführbare SIBs vollständig generiert. Bei der Benutzung dieses Tools hat sich mit der Zeit herausgestellt, dass dies oft in der Theorie besser funktioniert als in der Praxis. Zum einen nutzen die Anbieter von Web-Service-APIs in vielen Fällen ungewöhnliche Konstrukte, die von den Standard-Java-Stub-Generierungstools nicht verarbeitet werden können, zum anderen erhält man so grundsätzlich SIBs, welche die Struktur des Services exakt abbilden. Man hat also keine Möglichkeit, das SIB benutzerfreundlicher zu machen. Das technische Problem der vielfältigen Web-Service-Lösungen wurde zum Teil dadurch angegangen, dass man bei WSDL2SIB eine Auswahl verschiedener Java-Webservice-Bibliotheken hat[31], so dass bei dem Versagen der einen eine

[26] R ist eine Implementierung der Spezifikation S und eine Programmiersprache für statistische Aufgaben.

[27] z. B. ein Biologe, der die Ergebnisse seiner Analysen oder Experimente statistisch auswerten möchte, selbst jedoch kein Statistiker ist

[28] kurz JRI, zum Aufruf lokaler R-Skripte

[29] eine Client-Server-Lösung zum Aufruf entfernt deployter R-Skripte

[30] Java Electronic Tool Integration, siehe dazu auch Abschnitt 2.6.4.4

[31] Axis: http://axis.apache.org/axis, Axis2: http://axis.apache.org/axis2/java/core und die Referenzimplementierung von JAX-WS: https://jax-ws.java.net/

Alternative eingesetzt werden kann. Das Problem wurde so jedoch höchstens vermindert, nicht beseitigt. In vielen Fällen führte dies dazu, dass letztendlich doch auf den Einsatz von WSDL2SIB verzichtet wurde und das SIB von Hand entwickelt wurde.

SIB-Generierung für ERP-APIs: Bei der Generierung von SIBs für die APIs von ERP-Systemen ist es besonders wichtig, dass diese SIBs einfach zu benutzen sind. Dazu müssen sie die richtige Granularität besitzen, gut organisiert und strukturiert sein und eine gute und vollständige Dokumentation besitzen.

Eine vollautomatische Generierung von SIBs kommt hier nicht in Frage, da dies bedeuten würde, dass die entstehende SIB-Sammlung eine exakte Kopie der dahinterliegeden API darstellen würde. Bei sehr unterschiedlich (und unterschiedlich gut) erfolgten Umsetzungen der APIs ist dies nicht gewünscht. Die SIBs sollten möglichst einheitlich zu benutzen sein, unabhängig vom dahinter liegenden System. Außerdem sollte das Vokabular bei SIB-Namen, Parameter-Namen, Branch-Namen und in der Dokumentation dem Vokabular des Benutzers entsprechen, nicht dem des API-Anbieters. Gleiches gilt für die Organisation und Kategorisierung der Services bzw. SIBs. All dies kann nur durch eine halb-automatische Generierung erreicht werden. „Halb-automatisch“ bezieht sich hier auf den gesamten Generierungsprozess, der mehrstufig aufgebaut ist:

1. Sammeln aller nötigen Informationen
   a) Lesen der Informationen aus der API (z. B. mit entsprechenden Parsern)
   b) Eingabe von Informationen durch den Benutzer in einem Wizard (der Benutzer wird also geführt)
2. Die eigentliche Generierung des Quelltextes

Der Begriff „halb-automatisch“ bezieht sich hier also lediglich um den zweiten Unterpunkt des ersten von zwei Teilen, bei dem der Benutzer eingreifen muss. Hier kann der Benutzer entscheiden, welche Parameter wie benutzt werden sollen, wie sie im SIB heißen sollen und wie genau die Dokumentation aussehen soll. Sowohl die Analyse der API also auch die eigentliche Generierung laufen voll-automatisch. Das Ergebnis ist dabei ein fertiges Generat, welches weder gelesen noch weiter manipuliert werden muss.

Damit dieser Generierungsprozess so durchgeführt werden kann, muss entsprechende Software existieren, welche die API analysiert, den Wizard steuert und schließlich den Quelltext generiert. Die Erstellung dieser Software stellt natürlich einen gewissen Aufwand dar. Dieser lohnt

sich immer dann, wenn die API eine gewisse Größe besitzt, was bei ERP-APIs typischerweise der Fall ist. Bei einer großen API spart man besonders deshalb viel Arbeit durch den Einsatz von Generierungstechniken, da immer wiederkehrende Aufgaben nur einmal (bei der Erstellung des Generators) erledigt werden müssen. Diese Aufgaben beziehen sich dabei oft auf technische Eigenheiten der APIs. Man muss also nur einmal mit bestimmten technischen Problemen kämpfen, nicht bei jedem SIB neu. Auch ist entscheidend, dass das Wissen über diese technischen Details beim Generator-Ersteller bleiben kann und nicht an all diejenigen kommuniziert werden muss, welche die Services mittels SIBs integrieren möchten. Die SIB-Generatoren für ERP-APIs sind also System-spezifisch und nicht technologiespezifisch wie z. B. beim WSDL2SIB-Generator des jETI-Plugins.

Als technische Grundlage zur SIB-Generierung kann sehr gut die Template-basierte Code-Generierung eingesetzt werden. Templates sind relativ einfach zu erstellen und zu manipulieren. Außerdem sind sie sehr gut wiederverwendbar. In Abschnitt 5.3.1 wird darauf eingegangen, welche Vorzüge dieser Ansatz konkret hat und welche spezifische Technologien für diese Arbeit gewählt wurden.

Die Kommunikation mit dem Benutzer kann durch einen so genannten „Wizard“ erfolgen. Der Benutzer wird dabei durch einen strikt vorgegebenen Prozess geführt. Hierauf wird in Abschnitt 5.3.2 genauer eingegangen.

Von besonderer Wichtigkeit ist Dokumentation der Services und SIBs. Nur mit einer guten und vollständigen Dokumentation in ein SIB auch gut benutzbar. Dokumentationstext sollte dabei immer möglichst überall zugänglich sein, wo Interesse daran bestehen könnte, diese zu lesen. Das kann zum Beispiel in einer Übersicht der SIBs in einem Repository oder auch direkt im Prozessmodell der Fall sein. Ist Dokumentation bei einem Service vorhanden, so muss beim Generierungsprozess unbedingt darauf geachtet werden, dass diese Information nicht verloren geht. Oft existiert Dokumentation und das Problem besteht lediglich darin, dass man diese nicht oder nur schlecht findet. Ist die Dokumentation zum Beispiel nur auf einer bestimmten Webseite oder gar nur in speziellen Handbüchern (als PDF oder sogar gedruckt) vorhanden, so ist sie nur schwer zugänglich. Ist sie jedoch Teil der API[32], so kann dieser Text direkt weiter benutzt werden. Bei Java-Stub-Generatoren wie wsimport von JAX-WS oder wsdl2java von Apache Axis wäre es ein leichtes, diese Texte auszulesen und sie als JavaDoc an die generierten Methoden zu hängen. Leider wird dies nicht getan. Im SIB-Generierungs-Wizard muss vorhandene und zugängliche Dokumentation grundsätzlich dem Benut-

[32] z. B. in den `documentation`-Tags eines WSDL-Dokumentes

zer präsentiert werden, welcher dann entscheiden kann, ob er diese Dokumentation unverändert als SIB-Dokumentation behält oder (was meistens besser ist) noch Teile hinzufügt und löscht, so dass es genau auf das SIB passt. Eine vollautomatische Generierung der SIB-Dokumentation wird hier nicht versucht zu erreichen.

In verschiedenen Projekten wurden mit den genannten Mitteln verschiedene SIB-Generierungstools erstellt, die allesamt Wizards für die Dateneingabe und Templates für die Generierung einsetzten. Die Diplomarbeit von David Karla ist Gegenstand von Abschnitt 5.3.3, eine ähnliche Diplomarbeit zur Integration von Intuit Quickbooks wird in Abschnitt 5.3.4 behandelt.

Die Erfahrungen, die in beiden Diplomarbeiten gewonnen wurden, wurden genutzt zur Entwicklung eines allgemeinen, einheitlichen Frameworks zur Generierung von SIBs für Business-APIs. Dieses Framework mit dem Namen „InBuS“ wird in Abschnitt 5.3.5 behandelt. In Abschnitt 5.3.6 wird die konkrete Integration von Microsoft Dynamics NAV mittels InBuS beschrieben und in Abschnitt 5.4 wird die Verwendung von InBuS schließlich an einem Beispiel illustriert.

5.3.1 Template-basierte Code-Generierung

Es existieren diverse unterschiedliche Technologien zur Generierung von Quelltext (oder Text im Allgemeinen). Eine naive Möglichkeit wäre die manuelle Implementierung in einer beliebigen Programmiersprache mit Hilfe von String-Konkatenationen. Dieser Ansatz sehr mächtig, da die gängigen Programmiersprachen Turing-vollständig sind. Andererseits ist der resultierende Code nur sehr schwer wartbar und voller technischer Fallstricke z. B. beim so genannten „Escaping“ von Sonderzeichen, die in der verwendeten Programmiersprache nicht innerhalb von Strings angegeben werden dürfen.

Neben diesem sehr einfachen Ansatz existieren noch viele weitere, z. B. Regel-basierte Ansätze, bei denen die Generierung durch die Angabe einer Menge von Transformationsregeln definiert wird. Dieser Ansatz wird zum Beispiel von XSLT[33] verfolgt. Dabei werden Regeln angegeben, wie aus einem XML-Dokument ein Textdokument eines beliebigen Zielformats generiert wird.

Ein weiterer Ansatz ist der Einsatz von „Templates“[34] und einer dazugehörigen „Template-Engine“. Ein Template ist vergleichbar mit einem Lückentext, also einem Text, bei dem bestimmte Begriffe durch Platzhalter ersetzt wurden. Dieses Prinzip ist ebenfalls von der Erstellung von

[33]XSLT: Extensible Stylesheet Language Transformation
[34]zu deutsch „Vorlage“

Serienbriefen bekannt, wie es zum Beispiel die Textverarbeitung Microsoft Word anbietet. Möchte man ein Zieldokument erstellen, so müssen diese Platzhalter (wie beim Serienbrief) durch entsprechende Werte ersetzt werden. Dies wird durch die Template-Engine übernommen, einer Software, die aus einem Template und einer gegebenen Wertebelegung für die Variablen (also die Platzhalter) ein Zieldokument generiert. Aus „`$name$, $strasse$, $plz$ $ort$"` wird also durch Anwendung der Template-Engine und einer entsprechenden Wertebelegung die Adresse „`Max Mustermann, Hauptstraße 1, 12345 Musterstadt`“.

Meist bieten Template-Engines deutlich mehr Features an als die reine Ersetzung von Platzhaltern. Zu diesen gehören Bedingungen und damit verbundene Entscheidungen, Schleifen oder sogar Wertezuweisungen oder Berechnungen. Manche Template-Sprachen sind so komplex, dass sie sogar Turing-vollständig sind. Neben bekannten Template-Engines, die zur Generierung von Webseiten eingesetzt werden (z. B. ASP[35] oder JSP[36]) ist zum Beispiel Apache Velocity[37] eine Template-Engine mit entsprechend mächtiger Template-Sprache. Für jemanden, der es gewohnt ist zu programmieren, ist eine solche Sprache recht einfach zu erlernen. Velocity wurde zum Beispiel in der Diplomarbeit von David Karla eingesetzt um SIBs für die SAP-BAPI zu generieren. Auf diese Arbeit wird in Abschnitt 5.3.3 genauer eingegangen.

5.3.2 Wizards

Sowohl die Arbeiten von David Karla (siehe Abschnitt 5.3.3) und Andre Ackermann (siehe Abschnitt 5.3.4) also auch das InBuS-Framework (siehe Abschnitt 5.3.5) setzten auf den Einsatz eines Wizards zur Kommunikation mit dem User vor der SIB-Generierung. Ein Wizard[38] ist eine bestimmte Art der graphischen Benutzeroberfläche, bei der ein Benutzer durch eine vorgeschriebene Folge von Schritten geführt wird. In jedem einzelnen Schritt ist nur wenig zu tun und es ist somit immer relativ klar, was vom Benutzer erwartet wird. Dies ist der große Unterschied zu einer gewöhnlichen Benutzeroberfläche mit Menüs, Buttons und vielen weiteren komplexen Elementen. Bei einem Wizard stellt sich für den Benutzer nie die Frage, wo er jetzt in der GUI suchen muss, um etwas bestimmtes zu erreichen. Stattdessen ist das Ziel von vornherein vorgegeben und eng mit dem Wizard verknüpft. So kann der Benutzer quasi „an die Hand

[35] ASP: Active Server Pages, Webtechnologie von Microsoft, heißt in der aktuellen Version ASP.NET und ist Teil des .NET-Frameworks
[36] JSP: Java Server Pages, Java-basierte Webtechnologie von Oracle
[37] http://velocity.apache.org/
[38] manchmal auch „Assistent“ genannt

genommen“ werden und Schritt für Schritt zum gewünschten Ergebnis geleitet werden.

Dryer [Dry97] sieht einen Wizard als eine Form eines „UI-Agents“. Ein UI-Agent wird hier folgendermaßen definiert:

> „User interface (UI) agents are new intelligent user interface technologies that can help prevent people from making mistakes by guiding them through information system tasks.“

Es geht bei einem UI-Agent demnach hauptsächlich um Fehlervermeidung durch eine strikte Führung des Nutzers.

Die zweite Variante eines UI-Agents ist nach Dryer ein „Guide“, welcher zusätzlich zu einer „gewöhnlichen“, komplexen GUI eingesetzt wird, das Nutzerverhalten beobachtet und an den passenden Stellen den Nutzer mit Ratschlägen behilflich ist. Ein bekanntes Beispiel ist hier zum Beispiel der Office-Assistent von Microsoft Office (auch als „Karl Klammer“ - im Englischen „Clippy“ - bekannt). Ein solcher Guide benötigt also immer ein gewisses Maß an künstlicher Intelligenz, da er zunächst selbst feststellen muss, was der Benutzer plant. Es ist allgemein bekannt, dass dies nicht immer besonders gut funktioniert oder das Erscheinen des Guides einfach in vielen Situationen unerwünscht ist. Bei der SIB-Generierung steht das Ziel eindeutig fest. Hier sollte demnach kein Guide, sondern besser ein Wizard eingesetzt werden.

Ein Wizard wird von Dryer folgendermaßen definiert:

> „... provide task assistance by breaking the task into a (typically) linear series of steps and presenting the steps to a person one at a time.“

Weiterhin wird hier ausgesagt, dass sich Wizards aufgrund ihrer linearen Struktur am besten für Aufgaben eignen, die ein durch einen Algorithmus erzeugtes Ergebnis besitzen. Künstliche Intelligenz kommt bei Wizards (im Unterschied zu Guides) nicht zur Verwendung. Bekannt sind Wizards vor allem von Software-Installationsprogrammen. Auch hier steht das Ziel („Ich möchte die Software installieren.“) vorher eindeutig fest, Details (Speicherort, Feature-Auswahl, ...) müssen jedoch noch abgefragt werden. Dieses Beispiel zeigt auch die potentielle Einfachheit der Bedienung. In vielen Fällen wird eine Software durch eine blinde Betätigung der Buttons „weiter, weiter, weiter, ..., fertig“ installiert. Somit wird einfach immer der vorgegebene Standard gewählt. Für den Großteil der Benutzer ist dies genau das gewünschte Verhalten. Man hat also erreicht, dass die Bedienung fast immer sehr einfach und nur in Ausnahmefällen etwas aufwendiger ist.

In dieser Arbeit wurde eine Java-Swing-basierte Wizard-Library aus dem Netbeans-Projekt von Sun Microsystems verwendet[39]. Diese bietet viele Funktionen, die alle Wizards gemeinsam haben:

- An der linken Seite befindet sich ein großes Feld mit allen Schritten und der Markierung, wo man sich aktuell befindet.
- Am unteren Rand befinden sich die typischen Buttons für Weiter, Zurück, Fertig.
- Man kann sich in einem Kontext Daten merken, die vom Benutzer eingegeben werden.
- Der letzte Schritt des Wizards ist der so genannte „Finisher". Hier muss die Generierung angestoßen werden. Dies ist wiederum ein mehrschrittiger Prozess, weshalb währenddessen ein Fortschrittsbalken angezeigt wird.

5.3.3 Integration der SAP-BAPI

Nach der in Abschnitt 5.2.2 beschriebenen Entwicklung von SIBs für die SAP-BAPI wurde schnell klar, dass die manuelle Entwicklung der SIBs immer wieder sehr ähnlich abläuft. Bei der extremen Größe der BAPI bedeutet dies demnach, dass regelmäßig sehr viel Arbeit mehrfach erledigt wird. Durch die Gleichförmigkeit der API bietet es sich also an, diesen Prozess zu automatisieren. In der Diplomarbeit von David Karla wurde daher ein SIB-Generierungstool für die SAP-BAPI entwickelt, basierend auf den oben beschriebenen Konzepten, nämlich zum einen des Einsatzes eines Wizards, zum anderen der Template-basierten Code-Generierung. Als Template-Engine wurde hier Apache Velocity[40] eingesetzt.

Als Grundlage diente hier die in 5.2.2 beschriebene SAP-Lib, die noch etwas erweitert wurde. So musste der Code nicht für die BAPI (also JCo) direkt, sondern für die SAP-Lib generiert werden, was den Code deutlich vereinfacht und kürzer hält.

Auf Basis der SAP-Lib wurde dann zunächst eine Sammlung statischer, manuell implementierter, technischer SIBs erstellt, die Grundfunktionen bereitstellen, z. B. zum Aufbau und Abbau der Verbindung zu einem SAP-ERP. Im Unterschied zu z. B. SOAP-Webservices, bei denen bei jedem Aufruf einzeln eine Authentifizierung stattfindet, muss bei der Benutzung der BAPI über JCo zunächst eine Verbindung aufgebaut werden, dann kann diese Verbindung für beliebig viele Service-Aufrufe genutzt werden und schließlich muss die Verbindung wieder abgebaut

[39] https://wizard.dev.java.net/ bzw. https://today.java.net/pub/a/today/2006/02/28/using-wizard-api.html
[40] http://velocity.apache.org/

werden. Ein alternativer Ansatz zur Bereitstellung expliziter SIBs zum Aufbau und Abbau der Verbindung wäre es gewesen, in jedem SIB die Verbindung auf- und wieder abzubauen. Dies wäre zwar sehr ineffizient, würde jedoch zu einem einfacheren Prozessmodell führen. Ein Kompromiss zwischen beiden Ansätzen ist es, dass ein SIB prüft, ob eine Verbindung vorhanden ist und bei bestehender Verbindung diese nutzt und danach bestehen lässt. Ist keine Verbindung vorhanden, so baut das SIB seine eigene Verbindung auf.

Pro Wizard-Durchlauf, also pro zu integrierendem Service, werden bis zu drei SIBs generiert:

- Ein SIB, welches eine Eingabe-GUI anzeigt (optional)
- Ein SIB, welches den Service aufruft (verpflichtend)
- Ein SIB, welches eine Ausgabe GUI anzeigt (optional)

Diese drei SIBs kommunizieren miteinander über den Ausführungskontext. Dazu werden spezielle Klassen des SAP-Plugins benutzt. Nach dem Service-Aufruf werden z. B. Objekte der Klasse `ReturnParameter` in den Kontext gelegt, welche die Rückgabedaten der BAPI-Funktion enthält. Diese Klasse hat verschiedene Unterklassen für die einzelnen Datentypen wie z. B. Char, Table, Structure oder Table.

Ein vom Diplomanden selbst erkanntes Manko des Wizards ist, dass immer wieder die Situation vorliegt, bei der eine manuelle Implementierung einfacher wäre, da verschiedene Eingabeparameter den gleichen Wert bekommen müssen oder mit einem konstanten Wert belegt werden müssen. Diese Fälle wurden hier nicht vorgesehen. Das führt also zu generierten GUIs, in denen bei jedem Prozessdurchlauf der gleiche Wert eingetragen werden muss. In neueren Implementierungen (siehe Abschnitt 5.3.5) wurde dies entsprechend berücksichtigt.

Das SAP-Plugin wurde auch im Rahmen der schon in Abschnitt 5.2.3 erwähnten Projekte zur Vorlesung „Aktuelle Themen der Dienstleistungsinformatik“ eingesetzt, so dass dessen Bedienbarkeit und Nutzen durch Studenten evaluiert wurde. Die Erfahrungen hier haben gezeigt, dass das komplexe Object `ReturnParameter` im Kontext schwierig zu benutzen ist. Möchte man weiter mit den Daten arbeiten, muss man genau wissen, wie dieses Objekt aussieht, also quasi den Quelltext lesen. Besser wäre es, wenn erstens ein einfacheres Objekt im Kontext liegen würde und zweitens in der SIB-Dokumentation stehen, wie dieses Objekt strukturiert ist. Im Projekt hat dieser Missstand dazu geführt, dass die Studenten für ein bestimmtes Laufzeitdatum (eine Material-ID) nicht den Kontext benutzt haben, sondern die GUI als Anzeige, wo dann der Wert abgelesen und auf einen Zettel handschriftlich notiert wurde, wonach später dann diese ID wieder in ein Formular eingetragen wurde. Dies widerspricht

einem Grundgedanken von BPM, Prozesse möglichst zu automatisieren. In weiteren Wizards (siehe Abschnitt 5.3.5) wurde dies berücksichtigt.

5.3.4 Integration von Intuit Quickbooks

Nachdem mit der SAP-BAPI ein System betrachtet wurde, und evaluiert wurde, inwieweit eine automatische Generierung von SIBs möglich und praktikabel ist, wurde an der Universität Potsdam[41] von André Ackermann eine Diplomarbeit angefertigt, in der entsprechendes für das in Abschnitt 3.2.4 behandelte Intuit Quickbooks entwickelt wurde.

Dazu musste zunächst einmal erreicht werden, eine Kommunikation zwischen Java und Quickbooks herzustellen. Es wurde also eine entsprechende Klassenbibliothek konzipiert und in Java implementiert, die ähnlich wie die existierende QBFC-Bibliothek funktioniert, also die qbXML-Dateien parst bzw. generiert. Somit kann also nun von Java aus auf die Quickbooks-API zugegriffen werden, ohne dass im Java-Code XML manipuliert werden muss. Bei der Erstellung dieser Klassenbibliothek mussten mehrere Hürden genommen werden. Für qbXML existiert sowohl eine DTD als auch eine XSD-Spezifikation. Zunächst wurde die DTD untersucht. Eine Konvertierung der DTD nach XSD ist durch den Einsatz des Tools Apache Castor auch möglich. Es stellte sich heraus, dass die DTD sehr unvollständig war. Also wurde direkt die vorhandene XSD-Definition betrachtet. Mittels Apache Castor können daraus entsprechende Java-Klassen automatisch generiert werden. Apache Castor ist ein XML-Binding-Framework. Das bedeutet, dass man nach dem Generieren der Java-Klassen durch einen einzigen Methodenaufruf einfach die in XML kodierten Informationen als Java-Objekte instanziieren kann und auch umgekehrt, diese Java-Objekte als XML speichern kann. Ein Problem bei der Verwendung von Apache Castor waren Namenskonflikte innerhalb der XSD-Dateien. Diese waren jedoch mit vertretbarem Aufwand manuell zu beheben. Aber auch das um Namenskonflikte bereinigte XSD enthielt immer noch (wie die DTD) Fehler und Unvollständigkeiten. Um diese zu Beheben, wurde eine weitere Informationsquelle herangezogen. Die On-Screen-Reference, also die Webseite mit der API-Dokumentation, erhält ihre Informationen aus Dateien im JSON-Format. Diese erwiesen sich als qualitativ gut. Sie enthielten also keine Fehler und waren auch vollständig. Sie wurden also benutzt, um die XSD-Definition entsprechend zu erweitern.

Da die Kommunikation mit Quickbooks relativ schwierig ist (aufgrund der proprietären COM-Technologie), wurden verschiedene Connection-Manager und auch ein QBDE-Server entwickelt, eine Windows-spezifische

[41] in Zusammenarbeit mit der TU Dortmund

Java-Anwendung, mit der von außen über TCP/IP kommuniziert werden kann und die alle Anfragen an Quickbooks weiter leitet. Somit hat man plattformunabhängigen Zugriff.

Die grundsätzliche Art der generierten SIBs unterscheidet sich in dieser Diplomarbeit deutlich von den im Abschnitt 5.3.3 beschriebenen SAP-BAPI-SIBs. Hier ist für ein SIB, welches für einen speziellen Funktionsaufruf benötigt wird, lediglich der entsprechende Name der qbXML-Nachricht anzugeben. Die Generierung erfolgt dann vollautomatisch. Das hat zur Folge, dass in einem Batch-Durchlauf eine SIB-Sammlung für die gesamte Quickbooks-API erzeugt werden kann, und das ohne Eingreifen des Benutzers. Die resultierenden SIBs sind dabei jedoch sehr technisch und ohne weitere Hilfs-SIBs nicht zu benutzen.

Zusätzlich zu den generierten SIBs gibt es daher eine ganze Reihe manuell implementierter SIBs:

- InitQuickbooks (allgemeine Initialisierung)
- CheckConnectionTicket (Überprüfung des ConnectionTickets)
- CheckSessionTicket (Überprüfung des SessionTickets)
- GetSignOnTicket (zur Authentifizierung)
- InitRequest (Initialisierung der Anfrage)
- SelectRequest (Auswahl einer Anfrage)
- AddRequest (Hinzufügen einer Anfrage)
- SendRequest (Verschicken einer Anfrage)
- GetResponse (Auslesen der Antwort)
- SelectResponseItem (Auswahl eines Teiles der Antwort)

Auch diese sind insgesamt sehr technisch und kleinschrittig. Man hat es demnach insgesamt nur mit SIBs zu tun, die der Kategorie „technische Service-Aktivitäten“ (siehe Abschnitt 4.4) angehören. Konkrete Szenarien müssen im jABC als SLG zusammengestellt werden. Dazu werden die technischen SIBs entsprechend orchestriert. Man benötigt also einen ganzen SLG für einen Funktionsaufruf, bestehend aus relativ vielen SIBs. Dies ist der große Unterschied zum SAP-Plugin von David Karla, wo Business-Aktivitäten generiert wurden und zwar maßgeschneidert und On-Demand.

Da diese Modelle immer wieder sehr ähnlich aussehen und ein Business-Experte solche Modelle nicht selbst erstellen möchte bzw. kann, wurde zusätzlich der „Quickbooks Model Generator“ entwickelt. Dieser generiert die beschriebenen SLGs zum Zugriff auf Quickbooks-Services. Auch hier wird ein Wizard eingesetzt, welcher vergleichbar ist mit dem im SAP-Plugin. Somit nähert sich das Konzept insgesamt wieder an das

des SAP-Plugins an. Der entscheidende Unterschied bleibt, dass hier auch auf einer sehr technischen Ebene noch ein SLG benutzt wird. Im SAP-Plugin wurde dies komplett versteckt.

5.3.5 Das InBuS-Framework

Nach den Erfahrungen mit den unterschiedlichen ERP-APIs (siehe Kapitel 3), den manuellen SIB-Implementierungen (siehe Abschnitt 5.2 sowie den Diplomarbeiten von David Karla [Kar09] und André Ackermann [Ack10] mit ihren Ansätzen zur SIB-Generierung für SAP-BAPI (siehe Abschnitt 5.2.2) resp. Intuit Quickbooks (siehe Abschnitt 5.3.4) wurden die Ergebnisse genutzt um ein einheitliches Konzept zur Wizard-gesteuerten, Template-basierten Generierung von SIBs für Business-APIs aufzustellen und ein entsprechendes Framework zu implementieren. Dieses Framework trägt den Namen „InBuS“, was für „Integrating Business Software“ steht. Es fügt sich durch die Bereitstellung eines jABC-Plugins nahtlos in dessen graphische Benutzeroberfläche ein. Das bedeutet, dass man direkt aus dem jABC heraus Wizards starten kann, um SIBs für entsprechende Business-Services zu generieren.

Die Kerneigenschaften von InBuS sind also die folgenden:

1. Führung des Benutzers durch einen Wizard (siehe Abschnitt 5.3.2)
2. Generierung der SIB-Quelltexte durch Templates (siehe Abschnitt 5.3.1)

Neben diesen beiden Grundprinzipien gibt es noch zwei wesentliche Anforderungen, die eng miteinander zusammenhängen:

1. Leichte Erweiterbarkeit und Wiederverwendbarkeit von Komponenten durch modularen Aufbau
2. Universelle Anwendbarkeit für beliebige APIs und beliebige BPMS

Die Möglichkeit zur Erweiterung ist von besonderer Wichtigkeit. Es sollte möglichst einfach und schnell gehen, InBuS z. B. um neue ERP-Systeme zu erweitern. Außerdem ist das InBuS-Konzept nicht auf das jABC beschränkt. Da auch andere BPMS Prozesskomponenten besitzen, die prinzipielle Ähnlichkeiten zu den SIBs des jABC aufweisen (genaueres dazu folgt in Kapitel 6), ist es möglich, auch für diese Systeme die Prozesskomponenten zu generieren. Auf die Erweiterung des InBuS-Konzeptes auf alternative BPMS wird in Kapitel 7 eingegangen. Damit Erweiterungen sowohl auf API-Seite als auch auf BPMS-Seite möglich sind, besitzt InBuS einen modularen Aufbau, der für beide Seiten unabhängige Implementierungsmodule vorsieht. Die API-Seite wird dabei im Folgenden „Backend“ genannt, die BPMS-Seite „Frontend“. Wenn InBuS also um

ein Frontend-Modul erweitert wird, so kann es danach für ein weiteres BPMS eingesetzt werden, wird es um ein Backend-Modul erweitert, so wird eine weitere Business-API unterstützt. InBuS dient also quasi als zentraler Hub, der die Anzahl der Integrationsmodule zwischen Backends und Frontends von quadratisch[42] auf linear[43] reduziert.

Die Template-Engine: Als Template-Engine wurde für InBuS das aus dem AntLR-Projekt[44] stammende StringTemplate[45] [Par04] verwendet. Dabei handelt es sich um eine recht minimalistische Template-Engine, die zu einer sehr guten Trennung von Template und Logik führt. Anders als z. B. bei Apache Velocity erlaubt StringTemplate keine Wertezuweisungen an Variablen, beliebige Schleifen oder bedingte Verzweigungen. Unterstützt wird die Ersetzung von Platzhaltern durch Werte und der Funktionsaufruf. Letzteres wird intensiv genutzt, so dass auch aus einem Template heraus andere (Unter-)Templates aufgerufen werden können. Zur besseren Übersicht können dabei mehrere Templates zu Template-Gruppen zusammengefasst werden. Diese fassen mehrere Templates in einer Datei zusammen. StringTemplate ist bewusst keine Turing-vollständige Sprache und zwingt den Entwickler somit dazu, die Logik außerhalb des Templates anzusiedeln. Somit bleibt das Template selbst leicht verständlich.

Ein nettes Detail der StringTemplate-Sprache (z. B. im Vergleich zu Velocity) ist die sehr gute Unterstützung Komma-separierter Listen. Diese kommen in Programmcode natürlicherweise sehr häufig vor, z. B. bei den Parametern von Methoden. Da es hier im Allgemeinen um Service-Integration geht und im speziellen um die Integration meist sehr komplexer Services mit zahlreichen Parametern, ist dies besonders praktisch. So ist es bei Velocity nötig, dafür eine Schleife zu programmieren, welche die Parameter ausgibt und für jede Iteration ein Komma anzufügen. Dabei muss jedoch zusätzlich bei jeder Iteration überprüft werden, ob es sich um die letzte Iteration handelt. Ist dies der Fall, so muss die Ausgabe des Kommas ausgelassen werden. Bei StringTemplate wird die Produktion Komma-separierter Listen stattdessen durch eine Anwendung eines Sub-Templates auf alle Elemente einer Menge durchgeführt, wobei explizit eine Zeichenkette (in diesem Fall ein Komma) als Trenner angegeben werden kann.

Nachdem mit einem Template der Code generiert wurde, wird mit dem Code-Beautifier Jalopy[46] der Code formatiert, so dass er auch menschen-

[42] Anzahl der BPMS multipliziert mit der Anzahl der Business-APIs
[43] Anzahl der BPMS und Anzahl der Business-APIs werden addiert
[44] http://www.antlr.org/
[45] http://www.stringtemplate.org/
[46] http://jalopy.sourceforge.net/ bzw. http://notzippy.github.com/JALOPY2-MAIN/

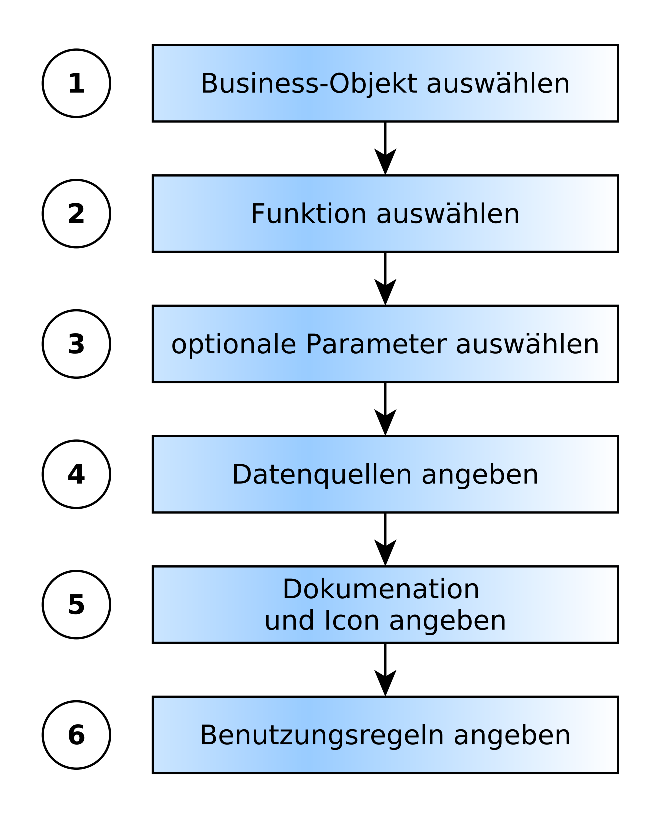

Abbildung 5.2: Die Schritte eines InBuS-Wizards

lesbar wird. In den Templates selbst muss daher nicht auf eine besonders schöne Formatierung des Resultats geachtet werden und es kann stattdessen eine besonders gute Lesbarkeit des Templates erreicht werden.

Der Wizard: Die einzelnen Schritte des Wizards sind in Abbildung 5.2 dargestellt.

Zunächst wird dabei das Business-Objekt ausgewählt, welches den gewünschten Service enthält. Die Darstellung der Business-Objekte geschieht dabei im Idealfall in hierarchischen Kategorien. Die Business-Objekte werden also als Baum[47] dargestellt. Die Informationen für diese Darstellung müssen aus einem entsprechenden Repository auf Backend-Seite gezogen werden. Je nach Backend ist dieser Schritt sehr unterschiedlich zu realisieren. Existiert kein Repository, so besteht für den Wizard-Ersteller die Möglichkeit, ein lokales Repository anzubieten. Dabei wird also bei InBuS selbst eine Sammlung der Business-Objekte und deren Kategorien gepflegt. Für den Benutzer sieht es dann so aus, dass er im ersten Schritt ein Repository sieht, welches nur schon benutzte

[47] in Java-Swing demnach als JTree

Business-Objekte enthält. Soll ein zusätzliches Business-Objekt angesprochen werden, so muss dies neu eingefügt werden und ist ab dem Zeitpunkt im lokalen Repository vorhanden. Kennt das Backend (also z. B. das Repository einer ERP-API) das Konzept der Business-Objekte nicht, kann der Benutzer die angebotenen Funktionen selbst manuell zu Business-Objekten zusammenfügen, was von da an ebenfalls lokal gespeichert wird. Bietet das Backend keine Kategorien an, so kann diese Zusatzinformation lokal gespeichert werden. Das lokale Repository speichert also all jene Information, die nicht im Repository des Backends vorhanden ist, und erweitert das Backend-Repository somit zu einem Repository mit in hierarchisch organisierten Business-Objekten, die wiederum jeweils zugehörige Methoden (also die Services) enthalten.

Im zweiten Schritt muss die konkrete Funktion ausgewählt werden, also eine Methode des vorher gewählten Business-Objektes. Dies ist der konkrete Service, der durch ein SIB im Prozessmodell repräsentiert und später zur Laufzeit aufgerufen werden soll.

In Schritt 3 wird eine Übersicht aller Ein- und Ausgabeparameter angezeigt und es muss für alle optionalen Parameter jeweils ausgewählt werden, ob diese im SIB angeboten werden oder nicht. Bei komplexen Datentypen kann es auch optionale Bestandteile des Parameters geben. Auch hier muss eine Auswahl durch den Benutzer vorgenommen werden. In dem Fall wird der Parameter samt seiner Bestandteile als Baum dargestellt, bei dem jeder Knoten eine Checkbox enthält.

Im vierten Schritt ist anzugeben, woher die Daten für die Parameter kommen. Dem Benutzer wird hier zunächst die Möglichkeit gegeben, Default-Werte anzugeben. Falls in einer Prozessausführung hier keine anderen Daten durch den Kontext geliefert werden, werden einfach diese Werte herangezogen. Der Standardweg der Datenquelle ist also der Ausführungskontext. Es wird also für Eingabeparameter ein entsprechendes Objekt im Kontext erwartet. Oft ist es jedoch auch so, dass man an bestimmte Parameter immer den gleichen Wert übergeben möchte, der also aus Sicht des Prozesses konstant ist. Dieser Wert kann also direkt vom Benutzer des Wizards festgelegt werden. Dieser Parameter taucht also im SIB gar nicht mehr auf. Ein Beispiel wäre ein Service, über den man Kontaktadressen anlegen kann, und der über einen bestimmten Parameter erfährt, ob es sich dabei um einen Kunden, einen Lieferanten oder einen Partner handelt. Nun können über diesen Mechanismus drei verschiedene SIBs erzeugt werden, einen für jede Art von Kontaktadresse. Manchmal ist es auch gewünscht, dass ein Wert an mehrere Parameter übergeben werden soll. Dies kann sowohl für einen konstanten Wert als auch für ein variables Objekt aus dem Kontext gelten. In dem Fall kann definiert werden, dass ein Eingabeparameter einfach seinen Wert von einem weiteren übernimmt.

In Schritt 5 ist die Angabe der Dokumentation und die Definition eines SIB-Icons möglich. Die Auswahl des Icons geschieht einfach über einen Standard-Filechooser. Es ist dabei möglich, sowohl Pixelgrafiken wie JPG, PNG oder GIF zu verwenden als auch Vektorgrafik wie SVG. Dokumentation muss für das SIB im Allgemeinen sowie für die einzelnen Parameter und Branches angegeben werden. Es gibt dabei verschiedene Quellen für die Dokumentation. Falls die zu Grunde liegende API Dokumentationstext ausliefert (z. B. in der WSDL), so kann diese als Vorschlag angeboten werden und vom Wizard-Benutzer direkt übernommen werden oder entsprechend abgewandelt werden. Bei den Parameter-Dokumentationen ist es besonders wichtig, dass zum einen annotiert wird, ob es sich um einen Ein- oder Ausgabeparameter handelt, zum anderen, wie die Struktur des Datentyps aussieht. Hier wird also eine Baumdarstellung der komplexen Datentypen mit angegeben. So ist es dem Prozessmodellierer besonders einfach möglich, zu sehen, wie die Daten aussehen müssen, die dem SIB bereitgestellt werden müssen und wie die Daten aussehen, die das SIB produziert.

Der sechste Schritt bietet schließlich die Möglichkeit, Benutzungsregeln anzugeben. Im Falle des jABC sind dies Regeln für den LocalChecker. Bestimmte LocalChecker-Regeln, die immer gelten werden hier nicht betrachtet. Diese werden immer in den Quelltext eingefügt. Hier geht es stattdessen um spezifische Regeln für diesen Service, also dieses SIB. Da LocalChecker-Regeln in Java definiert werden, wäre es denkbar, hier einfach nach Java-Quellcode zu fragen. Dies würde jedoch den Regeln der Einfachheit widersprechen und die meisten Wizard-Benutzer überfordern. Stattdessen werden bestimmte, vorgefertigte Regelarten angeboten, die genutzt werden können. So ist es grundsätzlich möglich, zu verbieten, dass bei einem SIB der Default-Wert einfach beibehalten wird. Wenn der Default-Wert zum Beispiel den Text „Nicht spezifiziert“ enthält, so kann durch diese Regel erreicht werden, dass auf keinen Fall vom Prozessmodellierer vergessen wird, hier einen gültigen Wert übergeben zu lassen. Bei Zahlen sind Regeln zum Wertebereich, wie ein Minimum, ein Maximum oder „Nicht 0“ möglich, bei Zeichenketten einfache Regeln wie „Nicht leer“ oder eine minimale Länge oder eine maximale Länge. Auch mittels regulärer Ausdrücke kann eine Zeichenkette überprüft werden. Dies erfordert schon deutlich mehr Kenntnisse vom Wizard-Benutzer, ist auf der anderen Seite jedoch auch sehr mächtig.

5.3.6 Integration von Microsoft Dynamics NAV

Am Beispiel von Microsoft Dynamics NAV wird im Folgenden beschrieben, wie mit dem InBuS-Framework SIBs für das jABC generiert werden. Wie in Abschnitt 3.2.3 beschrieben, existiert in Dynamics NAV für je-

des Business-Objekt eine so genannte „Page“, welche das Objekt samt seiner Daten sowie einer Reihe von Standardmethoden anbietet. Pages sind das Hauptmittel zur Organisation und zur Darstellung von Daten in Dynamics NAV. Jede Page bietet eine Reihe von Standardmethoden an, über die Daten erzeugt, gelesen, verändert und gelöscht werden können. Optional kann eine Page als „veröffentlicht“ markiert werden, was dazu führt, dass sie als SOAP-Webservice nach außen verfügbar gemacht wird. Im Auslieferungszustand ist zunächst keine Page veröffentlicht. Jede Page, die zugänglich gemacht werden soll, muss explizit so markiert werden. Außerdem muss ihr ein Name verliehen werden, über die sie extern verfügbar gemacht wird. Es existiert dabei ein WSDL-Dokument pro Page. Die URLs aller Page-WSDL-Dokumente werden zentral in einem DISCO-Dokument aufgelistet (siehe Abschnitt 3.2.3.1). Der vom Benutzer verliehene Name der Page ist dabei immer Teil der WSDL-URL. Außerdem ist durch das Vorkommen der Zeichenkette „Page“ an der URL direkt erkennbar, dass es sich um eine Page handelt und nicht um einen sonstigen Webservice von Dynamics NAV.

Die DISCO-Registry: Für den ersten Schritt des Wizards (siehe Abschnitt 5.3.5) ist es nötig, eine Liste aller Business-Objekte zu erlangen. Da Business-Objekte in Dynamics NAV den Pages entsprechen, wird also eine Liste aller veröffentlichten Pages benötigt. Genau dies steht im DISCO-Dokument. In Listing 5.3 ist zu sehen, dass fünf verschiedene Pages angeboten werden, die jeweils zum fiktiven Unternehmen „CRONUS AG“ gehören. Außerdem ist noch ein globaler Webservice „SystemService“ vorhanden, der allgemeine Informationen liefert, die für die hier angestrebte Integration der Services nicht benötigt werden. Auch ist hier zu sehen, dass keine weitere Kategorisierung der Pages vorgenommen wird. Dies ist aufgrund der relativ geringen Anzahl der Pages auch nicht unbedingt nötig. Neben Pages und dem SystemService existieren in Dynamics NAV noch so genannte „CodeUnits“, die ebenfalls als Webservice veröffentlicht werden können. Dabei handelt es sich um Skripte, die nicht mit einer bestimmten Page verbunden sind. Im Beispiel sind keine CodeUnits vorhanden. Für die meisten Integrationszwecke sind die Pages auch die deutlich elegantere Lösung. Mit CodeUnits kann der Benutzer eigene Skripte in Dynamics NAV implementieren, die dann intern sowie extern verfügbar gemacht werden können. Diese Form des Customizings führt jedoch oft zu einer schweren Wartbarkeit in der Zukunft. Aus diesem Grund werden CodeUnits im Folgenden nicht so intensiv wie Pages betrachtet.

Listing 5.3: Dynamics NAV - DISCO-Dokument

```
1 <discovery xmlns="http://schemas.xmlsoap.org/disco/"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
3     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4   <contractRef
5     ref="http://localhost:7047/DynamicsNAV/WS/CRONUS AG/SystemService"
6     xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
7   <contractRef
8     ref="http://localhost:7047/DynamicsNAV/WS/CRONUS AG/Page/customer"
9     xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
10  <contractRef
11    ref="http://localhost:7047/DynamicsNAV/WS/CRONUS AG/Page/employee"
12    xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
13  <contractRef
14    ref="http://localhost:7047/DynamicsNAV/WS/CRONUS AG/Page/itemcard"
15    xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
16  <contractRef
17    ref="http://localhost:7047/DynamicsNAV/WS/CRONUS AG/Page/salesorder"
18    xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
19  <contractRef
20    ref="http://localhost:7047/DynamicsNAV/WS/CRONUS AG/Page/vendorcard"
21    xmlns="http://schemas.xmlsoap.org/disco/scl/"/>
22 </discovery>
```

Um das DISCO-Dokument einzulesen musste für InBuS ein entsprechender Parser entwickelt werden. Da DISCO auf XML basiert und eine sehr einfache Struktur besitzt, wurde dies mittels JDOM[48] implementiert, einer Java-Implementierung eines DOM-Parsers. Der DISCO-Parser liest das Dokument ein und bietet im Wesentlichen folgende Informationen:

- Eine Liste aller enthaltenen URLs
- Die SystemService-URL
- Eine Liste der Namen aller Pages
- Eine Liste aller Page-URLs
- Eine Liste der Namen aller CodeUnits
- Eine Liste aller CodeUnit-URLs

Mit der Hilfe dieses Parsers stehen nun also die Namen aller Pages zur Verfügung und können auf der ersten Seite des Wizards zur Auswahl angezeigt werden. Wird eine Page ausgewählt, so steht die dazugehörige WSDL fest. Diese kann nun also heruntergeladen werden. Dazu wurde ein entsprechendes Download-Util implementiert, welches eine URL übergeben bekommt und den entsprechenden Inhalt der dahinter liegenden Datei als String zurückgibt. Dieses Tool kann dabei auch mit dem Microsoft-spezifischen NTLM-Protokoll zur Authentifizierung umgehen und wurde auch schon im ersten Schritt eingesetzt, um an das DISCO-Dokument zu gelangen.

Importieren des Webservices: Nach dem Herunterladen des WSDL-Dokumentes kann dies nun zur Generierung von Java-Stubs genutzt werden. In diesem Fall wurde dafür die Java-Referenzimplementierung von

[48]http://www.jdom.org/

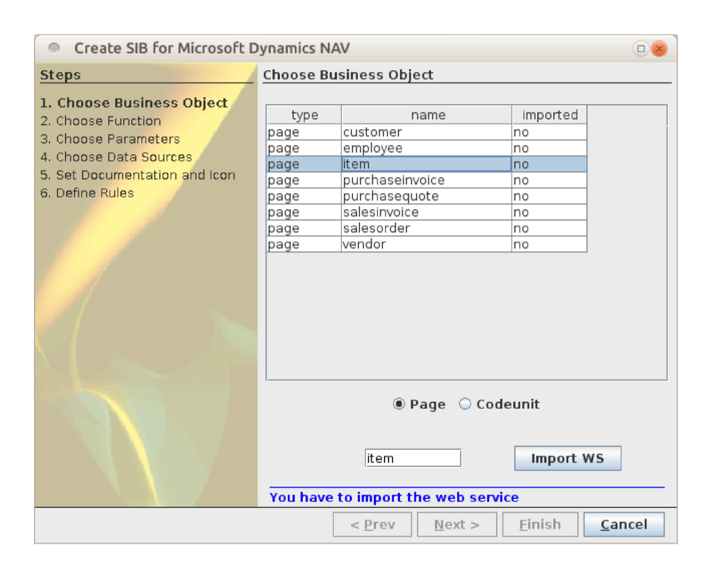

Abbildung 5.3: Auswahl der Page im InBuS-Wizard

JAX-WS eingesetzt. JAX-WS bietet für diesen Zweck das Kommandozeilentool `wsimport` an. Dieses ist in InBuS direkt eingebunden und wird vom Wizard direkt nach der Auswahl der Page und dem Herunterladen der WSDL ausgeführt. Die Bereitstellung der entsprechenden Java-Klassen wird hier als „Importieren“ des Webservices bezeichnet. Durch Java-Reflection kann zu jeder Zeit festgestellt werden, ob ein bestimmter Webservice bereits importiert wurde oder nicht. Nur wenn die Java-Klassen für die Page noch nicht existieren, wird der Importiervorgang ausgelöst. Ansonsten können die bestehenden Klassen genutzt werden. In Abbildung 5.3 ist die entsprechende Seite des Wizards dargestellt.

Auswahl der Operation: Die Informationen über angebotenen Operationen einer Page und die Struktur der Parameter kann am einfachsten direkt aus dem WSDL-Dokument ausgelesen werden. Man benötigt also einen Parser, der ein WSDL-Dokument einer Dynamics-Page oder CodeUnit einliest und folgende Informationen liefert:

- Wie ist der Name des Business-Objektes?
- Welche Operationen werden angeboten?
- Welche Parameter besitzen die Operationen?

Eine Recherche zu Open-Source-Implementierungen von WSDL-Parsern in Java hat ergeben, dass zwar viele Libraries zu Web-Services im Allgemeinen existieren, jedoch kein WSDL-Parser, der genau die oben angesprochenen Ergebnisse liefert. Vorhandene Lösungen lesen zwar intern auch WSDL-Dokumente ein, erzeugen daraus dann jedoch direkt Client-Stubs, ohne die erkannten Strukturen der WSDL öffentlich nach außen

hin anzubieten. Da eine WSDL eine sehr einfache Grundstruktur hat, war es jedoch nicht sehr aufwändig, hier einen entsprechenden Parser selbst zu implementieren. Auch hierfür wurde wieder JDOM verwendet. Zunächst wurde ein allgemeiner WSDL-Parser entwickelt, welcher noch nicht spezifisch für Dynamics NAV angepasst wurde. Dieser Parser kann natürlich immer wieder verwendet werden, auch bei anderen ERP-APIs, die WSDL einsetzen. Für Dynamics NAV wurden dann mit einem Page-WSDL-Parser und einem CodeUnit-WSDL-Parser zwei Spezialisierungen entwickelt, die auf die Eigenheiten von Pages resp. CodeUnits besonders eingehen. Der Page-WSDL-Parser bietet nach dem Einlesen des WSDL-Dokumentes eine Liste aller Operationen an sowie eine Liste aller Parameter an. Außerdem kann gezielt danach gefragt werden, ob die Page eine bestimmte Standard-Operation auch wirklich anbietet. Diese Informationen können genutzt werden, um alle verfügbaren Operationen zur Auswahl im Wizard anzuzeigen. Der Benutzer muss nur noch die gewünschte Operation auswählen.

Auswahl der Parameter: Nachdem die konkrete Operation feststeht, können nun alle für das SIB relevanten Parameter ausgewählt werden. Da die Parameter in Dynamics NAV grundsätzlich einfache (also keine komplex verschachtelten) Datentypen besitzen, ist hier keine Baumdarstellung nötig. Stattdessen reicht eine einfache Liste mit entsprechenden Checkboxen.

Weitere Wizard-Schritte Wie schon in Abschnitt 5.3.5 beschrieben und dort in Abbildung 5.2 dargestellt, folgen zum Schluss noch drei Schritte im Wizard, die Auswahl der Datenquellen für die Parameter (was meistens der Ausführungskontext ist), die Angabe von Dokumentation und Icon sowie die Definition von Benutzungsregeln.

SIB-Generierung Die Generierung des SIBs und aller damit zusammenhängenden Klassen (z. B. die Service-Adapter) wird im so genannten Finisher des Wizards angestoßen, also in der Prozedur, die durch die Betätigung des „Fertig"-Buttons gestartet wird. Da hier ein mehrschrittiger Prozess abgearbeitet wird und jeder Schritt eine gewisse Zeit in Anspruch nimmt, wird während dieser Prozedur ein Fortschrittsbalken angezeigt.

Die einzelnen Schritte des Finishers sind die Folgenden:

1. Generate PortUtil
2. Copy BusinessObject
3. Generate JavaService

4. Generate LightweightServiceAdapter
5. Generate ServiceAdapter
6. Generate SIB
7. Generate Provider-SIBs
8. Generate SIB-Documentation
9. Compile PortUtil
10. Compile JavaService
11. Compile LightweightServiceAdapter
12. Compile ServiceAdapter
13. Compile SIB
14. Create Summary

Die ersten Schritte generieren Quelltext, danach wird dieser entsprechend compiliert und zum Schluss wir eine kurzer zusammenfassender Text erzeugt, der dem Wizard-Benutzer als Ergebnis angezeigt wird. Schritt 2 ist eine triviale Sonderform der Generierung. Da hier quasi ein Template ohne dynamische Inhalte existiert, das Ergebnis demnach also immer gleich aussieht, reicht es hier, einen fertigen Quelltext einfach in das Zielverzeichnis zu kopieren anstatt etwas zu generieren. Es handelt sich dabei um die Klasse `BusinessObject`.

Die Klasse `BusinessObject` dient als Container für die Laufzeitdaten. Die von JAX-WS generierten Datenobjekte, die genau denen des Services entsprechen, können nicht einfach verwendet werden, da sie auch die Elemente enthalten, die im Wizard nicht ausgewählt wurden. Nun gibt es zwei Möglichkeiten, mit diesen Daten umzugehen: Entweder generiert man neue Klassen (z. B. JavaBeans) für die Datenobjekte oder man verwendet generische Datenstrukturen wie Abbildungen (`java.util.Map`) oder Listen (`java.util.List`). Bei InBuS wurde der zweite Ansatz gewählt. So wird der Generierungsaufwand niedrig gehalten und es existieren aus technischer Sicht immer einfach zu verstehende Datenstrukturen. Die Klasse `BusinessObject` erbt also von `java.util.Map`, wobei die Schlüssel die Namen der Parameter und die Werte die entsprechenden Werte der Parameter sind. Jeder Wert kann dabei wieder eine Abbildung oder eine Liste sein, und Listen können wieder Abbildungen oder Listen enthalten. So sind beliebige komplexe Datenstrukturen abbildbar. Zusätzlich existiert noch ein Sonderfeld für den Typ-Namen des Business-Objekts. Dieses Objekt wird also durch das SIB im Kontext gespeichert bzw. wird dort vor seiner Ausführung erwartet. Bei der Eingabe reicht auch eine einfache Map, hier ist die Benutzung der Klasse `BusinessObject` nicht zwingend erforderlich. Dieses Datenobjekt kann

danach von weiteren SIBs besonders einfach benutzt werden. Bei einem manuell implementierten SIB ist der Zugriff auf eine Abbildung oder Liste immer möglich, aber auch bestimmte fertige SIBs können gut damit umgehen. Viele Common-SIBs arbeiten zum Beispiel mit der Expression-Language von Java-EE. In dieser kann durch eine sehr einfache Syntax auf die Daten zugegriffen werden. Liegt zum Beispiel unter dem Schlüssel `customer` ein Kundenobjekt im Kontext, und das entsprechende Objekt besitzt die Felder `name` und `city`, so kann auf diese Daten einfach mittels `customer.name` bzw. `customer.city` zugegriffen werden. Liegt unter `customers` eine Menge von Kunden, so kann auf den ersten mittels `customers[0]`, auf dessen Namen mittels `customers[0].name` zugegriffen werden. Durch die Darstellung der Struktur des Business-Objektes in der SIB-Hilfe kann ein Modellierer oder Entwickler immer sehr einfach nachschauen, über welche Bezeichner auf bestimmte Daten zugegriffen werden muss.

Bei der Entwicklung von InBuS war stets wichtig, dass die generierten SIBs auch mit beliebigen anderen SIBs kombinierbar sind, egal ob sie mit einem anderen Tool generiert wurden oder von Hand programmiert wurden. Daher wurde zum Beispiel darauf verzichtet, ein spezielles globales Datenmodell einzuführen, das alle SIBs benutzen müssen. Stattdessen steht jedes SIB mit seinen Daten für sich und diese Daten können beliebig erzeugt, gelesen, transformiert oder gelöscht werden. Praxiserfahrungen haben gezeigt, dass komplexe Transformationen oft durch manuell implementierte SIBs am einfachsten zu realisieren sind. Für einfache Zwecke (wie z. B. die Extraktion eines Elementes aus einem Business-Objekt) bietet sich die Verwendung der erwähnten Common-SIBs an, die auf die Verwendung der Expression-Language setzen. Für die Erzeugung der Daten (z. B. für die Eingabedaten des Services) kann ein Konzept wiederverwendet werden, welches durch das jETI-Framework [Kub13] eingeführt wurde: Die Generierung so genannter „Provider-SIBs“. Dabei handelt es sich um SIBs, die die benötigten Datenobjekte zusammensetzen und in den Kontext legen. Dazu muss ihnen als Eingabe die Menge aller Bestandteile übergeben werden. Ein Provider-SIB für das erwähnte Kundenobjekt würde demnach jeweils einen Namen und eine Stadt an definierten Stellen im Kontext erwarten, das Kunden-Objekt daraus erzeugen und es danach in den Kontext legen.

Die Klasse `JavaService` ist der Wrapper um den von JAX-WS generierten Web-Service-Aufruf, der den Service als Java-Methode anbietet. Zusammen mit den JAX-WS-Klassen und der Klasse `BusinessObject` bildet sie das Backend, also den Teil, der spezifisch für die zu Grunde liegende Business-API ist. Alle anderen Generate, also der ServiceAdapter, der LightweightServiceAdapter und das SIB bilden das Frontend. Sie sind spezifisch für das jABC.

Die SIB-Dokumentation wird in eine separate Datei geschrieben. Technisch gesehen handelt es sich dabei um eine Java-Properties-Datei, also eine Textdatei mit Schlüssel-Wert-Paaren, die jeweils in einer Zeile stehen und bei denen Schlüssel und Wert durch ein Gleichzeichen getrennt sind. In dieser Datei stehen sowohl der allgemeine Dokumentationstext des SIBs als auch die erklärenden Texte zu allen Parametern und Branches. Bei den Parametern ist zu beachten, dass auch die Struktur der komplexen Parametern in einer Baumstruktur angegeben werden. So ist für den SIB-Benutzer gut zu erkennen, wie er diese Daten nutzen oder bereitstellen kann.

Die generierten Provider-SIBs dienen zur Erzeugung der Eingabedaten. Da bei Dynamics NAV die komplexen Parameter jeweils wieder nur aus einfachen Datentypen bestehen (es auch keine weitere Verschachtelung komplexer Parameter gibt) und es pro Service genau einen Parameter gibt, entstehen hier also nur ein Provider-SIB mit vergleichsweise vielen Parametern - einem SIB-Parameter pro Bestandteil des komplexen Parameters. Würde der komplexe Parameter wieder einen weiteren komplexen Parameter enthalten, so müsste hierfür wieder ein entsprechendes Provider-SIB generiert werden.

Benutzung der generierten SIBs: Sind die SIBs fertig generiert, so können sie mit dem Taxonomie-Editor übersichtlich in einer hierarchischen Struktur angeboten werden. In Abbildung 5.4 ist zu sehen, wie diese Taxonomie in der graphischen Benutzeroberfläche des jABC dargestellt wird. Hier können diese SIBs nun einfach gefunden werden und bei Bedarf per Drag-and-Drop auf der Prozessfläche platziert werden. Wird die Taxonomie groß, so ist das Suchfeld praktisch, mit dem man unter der Benutzung von Wildcards die Sammlung der SIBs filtern kann.

5.4 Anwendungsbeispiel

Im Folgenden wird an einem kleinen Beispiel gezeigt, wie die Umsetzung eines Prozesses mit sehr verschiedenen Aktivitäten in jABC aussehen kann. In Abschnitt 5.4.1 wird dazu zunächst ein Szenario vorgestellt, welches umgesetzt werden soll. Danach wird in Abschnitt 5.4.2 beschrieben, wie dieses Szenario als Prozess im jABC modelliert wird und wie die dazugehörigen SIBs entstehen.

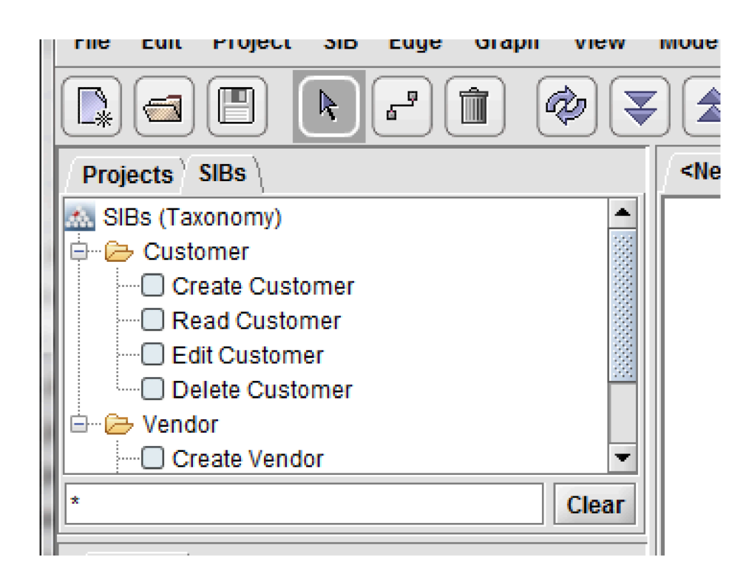

Abbildung 5.4: SIB-Taxonomie

5.4.1 Szenario

Folgendes Szenario zum Thema „Mitarbeitereinstellung“ soll umgesetzt werden:

In einer mittelständischen Unternehmensberatung, welche aufgrund der guten Marktlage in der nahen Zukunft mit einem enormen Anwachsen der Mitarbeiterzahl rechnet, soll die Einstellung neuer Mitarbeiter sobald wie möglich standardisiert und organisiert ablaufen. Bisher musste ein neuer Mitarbeiter sich oft selbst um viele Dinge aktiv bemühen und z. B. durch eigene Recherchen herausfinden, wen er ansprechen muss, damit er ein bestimmtes Recht in einem IT-System bekommt oder dass ihm ein bestimmtes Arbeitsmaterial (z. B. Computer-Hardware) ausgehändigt wird. Es war keine Seltenheit, dass bestimmte Punkte erst lange Zeit nach der Einstellung (oder sogar nie) erledigt wurden, da sie dem neuen Mitarbeiter schlichtweg unbekannt waren und sich von seinen Kollegen niemand zuständig fühlte, sich darum zu kümmern.

Nun soll der Einstellungsprozess definiert, modelliert und auf einer Engine zur Ausführung gebracht werden. Der Prozess soll dabei soweit wie möglich automatisiert werden. Effizienz und Nachvollziehbarkeit sind dabei von besonderer Wichtigkeit. Es soll also keine Mehrfacheingaben von Daten geben und alle involvierten Systeme sollen automatisch die benötigten Daten übermittelt bekommen. Durch den Einsatz einer Engine kann außerdem ein Monitoring durchgeführt werden und somit festgestellt werden, ob die Prozesse zügig durchlaufen oder es doch unerwünschte Verzögerungen gibt. Bei Bedarf kann dann der Prozess optimiert werden.

Interviews mit der Unternehmensführung, der IT-Abteilung und zahlreichen Mitarbeitern haben ergeben, dass der Prozess folgende Punkte umfassen muss:

- Jeder neue Mitarbeiter muss folgende Schulungen besuchen:
  - Allgemeine Einführung in das Unternehmen
  - Sicherheitsunterweisung
  - Einführung in die IT-Landschaft des Unternehmens
- Jedem Mitarbeiter muss ein Arbeitsplatz (Büro, Schreibtisch, Stuhl, ...) bereitgestellt werden.
- Jedem Mitarbeiter muss seine entsprechende Computer-Hardware ausgehändigt werden.
- Es müssen Visitenkarten gedruckt werden. Ein dafür geeigneter Drucker ist im Unternehmen selbst vorhanden.
- Der Mitarbeiter muss die korrekten IT-Rechte auf allen Servern im Unternehmensnetzwerk bekommen.
- Die Mitarbeiterdaten müssen im ERP-System eingetragen werden, da hierüber z. B. die monatliche Abrechnung erfolgt. Als ERP-System ist Microsoft Dynamics NAV im Einsatz.

5.4.2 Umsetzung

Bei der Umsetzung des in Abschnitt 5.4.1 beschriebenen Szenarios wird nun so vorgegangen, dass zunächst analysiert wird, welche Aktivitäten benötigt werden. Da hier eine Umsetzung mit dem jABC beschrieben wird, bedeutet dies also eine Definition der Menge der SIBs. Für jedes SIB muss dann entschieden werden, ob es schon vorhanden und in dieser Form direkt benutzbar ist oder ob es noch erstellt werden muss. In letzterem Fall muss dann weiter entschieden werden, auf welche Art und Weise das SIB erstellt werden soll, ob es zum Beispiel automatisch oder halbautomatisch generiert werden kann.

Um die Menge der benötigten SIBs zu identifizieren, ist es am einfachsten, zunächst den Prozess mit entsprechenden Platzhaltern zu modellieren, also mit SIBs, die zunächst einmal keine Funktion haben und erst später durch funktionsfähige SIBs ersetzt werden. In Abbildung 5.5 ist der Haupt-SLG[49] dargestellt. Das zweite SIB („Mitarbeiter schulen“ ist ein GraphSIB[50], referenziert also einen weiteren SLG, in diesem Fall den dreischrittigen SLG aus Abbildung 5.6.

[49]der SLG auf der obersten Ebene der Hierarchie
[50]erkennbar am grünen Punkt

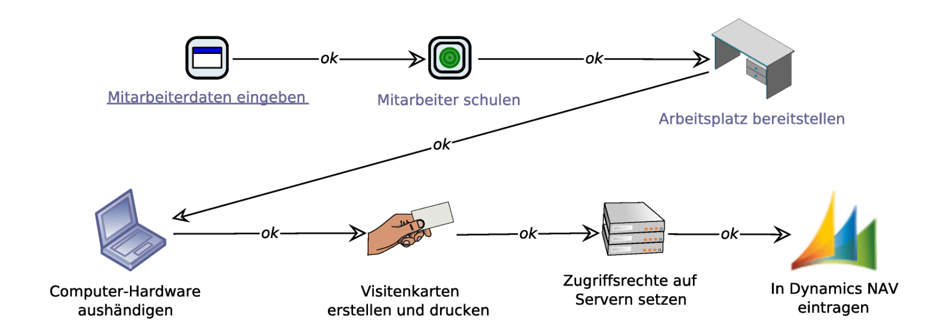

Abbildung 5.5: SLG: Einstellung eines neuen Mitarbeiters

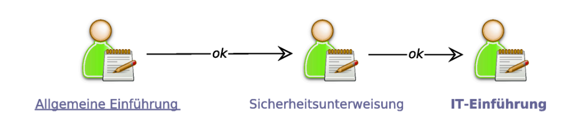

Abbildung 5.6: SLG: Untermodell - Mitarbeiter schulen

Das erste SIB („Mitarbeiterdaten eingeben“) bewirkt die Anzeige eines Formulars, in das die persönlichen Daten des Mitarbeiters (Name, Adresse, Geburtstag, ...) eingetragen werden. Das beste Ergebnis wird hier mit einem manuell implementierten Formular erzielt[51]. Alternativ kann auch ein GUI-Generierungsframework eingesetzt werden. Speziell für Formulare gibt es hier eine ganze Reihe von Tools[52]. Für einen einfachen Prototyp wäre hier sogar eine Reihe von einfachen Input-SIBs aus den Common-SIBs ausreichend, die jeweils in einem kleinen Dialog nach einem einzelnen Wert fragen. Diese Daten werden in jedem Fall im Ausführungskontext gehalten und können damit von allen weiteren SIBs benutzt werden.

Bei manuell implementierten SIBs ist dabei grundsätzlich ratsam, nur den wirklichen Implementierungsteil von Hand zu erstellen. Die Definition des SIBs kann besser, schneller, einfacher und weniger fehleranfällig mit der SIBCreator-GUI[53] erstellt werden.

Das zweite SIB referenziert wie erwähnt das in Abbildung 5.6 dargestellte Untermodell. Hier befinden sich drei manuelle Tasks, nämlich die

[51] Auf welche Anzeigetechnologie, also z. B. Java-Swing oder HTML, ist für dieses Beispiel unerheblich.

[52] Beispiele sind zu finden unter http://www.cmsfrog.de/form_generator/formular-generator.html oder http://code.google.com/p/swing-formbuilder

[53] siehe dazu auch Abschnitt 5.3

Durchführung von Schulungen. Hier sind keine technischen Services aufzurufen, sondern stattdessen durch Personen Tätigkeiten zu vollziehen. Es handelt sich also hier um „Human Tasks“, wie sie zum Beispiel auch von BPEL4People adressiert werden. Üblich ist hier eine Umsetzung mittels Aufgabenlisten. Jeder involvierte Benutzer hat seine eigene Liste von Aufgaben. Die Ausführung einer manuellen Aktivität im Prozess hat zur Folge, dass eine neue Aufgabe zu der entsprechenden Liste hinzugefügt wird. Hat der Besitzer der Aufgabenliste diese Aufgabe erledigt, so kann er sie entsprechend abhaken und der Prozess kann mit der Ausführung fortfahren. Es existieren zahlreiche Implementierungen von Aufgabenlisten. Eine bekannte Cloud-Lösung ist hier „Google Tasks“, eine einfache SaaS-Lösung, aus der Sammlung „Google Apps“, die auch schon in Abschnitt 5.2.3 wurden. Diese Cloud-Lösung besitzt eine Java-Bibliothek, so dass sie sehr einfach von einem SIB aus angesprochen werden kann.

Für den Druck der Visitenkarten sind im Unternehmen bereits Vorlagen im Format der Textverarbeitung von OpenOffice vorhanden. Es können hier also die fertigen OpenOffice-SIBs verwendet werden. In diesem Fall würden SIBs verwendet werden, welche OpenOffice starten, die Vorlage laden und entsprechende Platzhalter durch die echten Werte ersetzen. Auch das Speichern der Datei und der direkte Anstoß eines Druckauftrags ist mit den SIBs möglich. Für die Realisierung würde man also das Platzhalter-SIB durch ein GraphSIB ersetzen, welches die beschriebenen OpenOffice-SIBs enthält.

Für die zentrale Vergabe von IT-Rechten für alle Server im Unternehmensnetzwerk bietet sich der Einsatz eines Verzeichnisdienstes an. Dies kann zum Beispiel ein LDAP-Server[54] sein. Der Zugriff auf LDAP von Java aus geschieht über die JNDI-Schnittstelle[55]. Die benötigten Klassen sind in Java SE[56] enthalten[57]. Einen vereinfachten Zugang zu LDAP verspricht „Spring LDAP“[58]. Man kann hier also das Fazit ziehen, dass es gut möglich ist, ein SIB für den Zugriff auf LDAP zu implementieren. Nun ist es jedoch wichtig, welchen Charakter dieses SIB haben soll. Eine Möglichkeit ist eine technische Service-Aktivität, also ein generisches LDAP-SIB, welches hauptsächlich von Technikern parametrisiert werden kann, da man LDAP und dessen Eigenheiten kennen muss. Die zweite Möglichkeit ist ein spezielleres SIB, welches zwar in seiner Implementierung auch LDAP anspricht, nach außen hin jedoch eine domänenspezifische Business-Aktivität darstellt. In diesem Fall wäre das ein SIB, welches bestimmte IT-Rechte an Rollen oder Personen vergibt. Dieses

[54]LDAP: Lightweight Directory Access Protocol
[55]JNDI: Java Naming and Directory Interface
[56]Java SE: Java Standard Edition
[57]in den Packages `javax.naming.directory` und `java.naming.ldap`
[58]http://www.springsource.org/ldap

SIB kann einfach auch von Nicht-Technikern benutzt werden.

Das letzte SIB dieses Prozesses ist besonders interessant, da hier ein ERP-System angesprochen wird, konkret Microsoft Dynamics NAV (siehe Abschnitt 5.3.6). Dieses SIB kann also mit InBuS (siehe Abschnitt 5.3.5) halbautomatisch generiert werden. Dazu wird der Wizard entsprechend ausgeführt, in dem man die Page „Employee“ und dann dort die Operation „Create“.

Anschließend können die gewünschten Datenfelder ausgewählt werden, wonach festgelegt wird, dass die Daten aus dem Kontext gelesen werden sollen. Hier ist zu bedenken, dass im Kontext eine einfache Map erwartet wird, die als Schlüssel die Parameternamen und als Werte die Parameterwerte beinhaltet. Dieses Vorgehen hat den Vorteil, dass keine neuen Klassen generiert und benutzt werden müssen. Nun gibt es verschiedene Herangehensweisen, die Map bereitzustellen. In diesem Fall ist es möglich, dass bei der Implementierung des ersten SIBs (die GUI zur Dateneingabe) direkt die gewünschte Datenstruktur erzeugt wird. Alternativ kann das GUI-SIB auch alle Datenfelder als einzelne Elemente mit jeweils einzelnen Schlüsseln in den Kontext schreiben. In dem Fall ist ein zusätzliches Provider-SIB (oder bei einer tieferen Verschachtelung mehrere Provider-SIBs) nötig, um die gewünschte Datenstruktur zu konstruieren.

Zum Schluss besteht noch die Möglichkeit, Dokumentationstext zu verfassen, ein Icon zu vergeben[59] und Benutzungsregeln (für den LocalChecker) festzulegen.

[59] standardmäßig das Logo von Dynamics NAV, was aber auch gegen jedes andere Icon ausgetauscht werden kann

KAPITEL 6

Service-Integration in anderen BPMS

Nachdem in Kapitel 5 dargestellt wurde, wie in jABC Services in die Prozesse integriert werden, befasst sich diesen Kapitel nun mit einer Auswahl alternativer BPMS. Es sind jeweils reine BPMS, also Systeme die sich mit dem Management von Geschäftsprozessen im engeren Sinne beschäftigen. Jedes Produkt besitzt dabei immer mindestens ein Prozess-Modellierungstool sowie eine Ausführungsumgebung (oder „Engine"). Es steht also jeweils der ausführbare Prozess im Mittelpunkt. Reine Modellierungstools wurden nicht betrachtet, da hier nicht der Bedarf der Service-Integration besteht. Ebenfalls wurden keine Integration-Frameworks ohne direkten Prozessbezug (wie z. B. ein ESB - Enterprise Service Bus) untersucht.

6.1 jBPM

Das Prozess-Framework jBPM[1] [Red13] ist ein Projekt der Firma JBoss, welche heute Teil des Unternehmens RedHat Software ist. Bis zur Version 4 wurde dabei eine eigene, proprietäre Prozessdefinitionssprache namens jPDL[2] eingesetzt. Wie die Namen von Softwareprodukt und Sprache schon andeuten, handelt es sich bei jBPM um ein Java-basiertes Framework mit einer entsprechenden Java-API um Geschäftsprozesse auszuführen und zu überwachen. Ein in jPDL definierter Prozess besteht aus

[1]jBPM: Eine Akronym für „Java Business Process Management"
[2]jPDL: Abkürzung für „Java Process Definition Language"

einer Prozessbeschreibung in XML sowie Service-Implementierungen in Java. Dies ist im Wesentlichen im jABC genauso. Bei jPDL rückt das XML jedoch deutlich mehr in den Mittelpunkt, da es hier sehr üblich ist, dies manuell zu bearbeiten. Es existiert zwar ein graphischer Editor, da dieser jedoch nicht besonders ausgereift und feature-reich ist, wird er somit eher als optionales „Add-on“ gesehen. Beim jABC wird XML nur als reines Serialisierungsformat gesehen, was grundsätzlich vor dem Anwender versteckt wird.

Im Mai 2011 verließen die Hauptentwickler von jBPM die Firma RedHat und starteten als Angestellte der DMS-Firma[3] Alfresco[4] ein neues Projekt namens „Activiti“, welches eine direkte Weiterentwicklung von jBPM 4 darstellt. Um dies zu betonen, besitzt die erste Version von Activiti die Versionsnummer „5.0“. Activiti wird in Abschnitt 6.2 behandelt. JBoss selbst führte die Entwicklung von jBPM 4 nicht weiter, sondern ließ künftige Entwicklungen auf Basis eines anderen JBoss-Produktes namens „Drools Flow“ entstehen. Dies ist wiederum Teil des Business-Rules-Frameworks „Drools“ [5]. Die technologische Basis des aktuellen BPMS von JBoss, genannt „jBPM 5“ ist also nicht jBPM 4 sondern „Drools Flow“.

Der größte Unterschied zwischen jBPM4 und seinen beiden Nachfolgern ist die benutzte Sprache. Im unterschied zu jBPM 4, bei der die proprietäre Sprache jPDL zum Einsatz kam, nutzen jBPM 5 und Activiti BPMN 2. Die Gründe, auf BPMN zu wechseln sind im Wesentlichen folgende:

- BPMN ist ein offizieller Standard der OMG und hat sich mittlerweile zu einem de-facto Standard im BPM-Bereich entwickelt
- Seit Version 2.0 deckt die Spezifikation nicht mehr nur die reine Notation ab, sondern enthält auch die Beschreibung einer Ausführungssemantik sowie eines XML-basierten Speicher- und Austauschformats.

In der Theorie sollten Diagramme in BPMN 2 demnach ausführbar und zwischen verschiedenen Produkten austauschbar sein. Unsere praktische Erfahrung mit verschiedenen BPMN-2-Produkten hat gezeigt, dass diese sich nicht wirklich immer an den Standard halten. Typischerweise wird immer nur eine Untermenge von BPMN 2 unterstützt, welche dann wiederum durch proprietäre Erweiterungen ergänzt wird. Auch die in der Spezifikation angestrebte Art der Service-Integration wird nicht immer genau so durchgeführt.

[3]DMS: Document Management System
[4]http://www.alfresco.com/
[5]http://www.jboss.org/drools/

6.1.1 jBPM 4.x (jPDL)

Wie schon weiter oben erwähnt, nutzt jBPM 4 jPDL als Prozessbeschreibungssprache. Es existieren zwei Kategorien von Aktivitäten in jPDL:

1. „Control Flow Activities“ und
2. „Automatic Activities“

„Control Flow Activities“ sind Aktivitäten, die den Kontrollfluss definieren oder beeinflussen, wie z. B. Start, End oder Decision. Die einzige „Control Flow Activity“ welche für den Aspekt der Service-Integration von Belang ist, ist die „Custom Activity“. Eine solche Aktivität beeinflusst zwar nicht zwingend den Kontrollfluss, da dies jedoch möglich ist, wird diese Aktivität zur ersten Kategorie gezählt.

Die „Automatic Activities“ sind einfache Aktivitäten, die lediglich ausgeführt werden, den Kontrollfluss jedoch nicht beeinflussen. Hier enthalten sind zum Beispiel eine Skript-Aktivität sowie mehrere Aktivitäten, die in dieser Arbeit in die Kategorie der technischen Aktivitäten fallen.

Skript-Aktivitäten Die Skript-Aktivität in jPDL interpretiert ein gegebenes Skript einer beliebigen Skript-Sprache mit einer zu JSR-223 kompatiblen Scripting-Engine[6]. Die Standardsprache ist hier jUEL, eine Implementierung der „Unified Expression Language“[7], einer recht einfach gehaltenen Ausdruckssprache, die im JavaEE-Standard definiert wird.

Technische Service-Aktivitäten In jBPM kann auf eine Auswahl mehrerer technischer Service-Aktivitäten zugegriffen werden. Die wahrscheinlich wichtigste ist die Java-Aktivität, welche eine externe Java-Methode aufruft. Hier kann entweder ein vollständiger Java-Klassenname (incl. Package) oder eine Expression definiert werden. Zusätzlich muss der Name der aufzurufenden Methode, eine Liste der Parameterwerte[8] und ein Kontextvariablenname für den Rückgabewert der Methode angegeben werden. Falls ein Klassenname definiert ist, wird die dazugehörige Klasse instantiiert und die Methode wird aufgerufen. Die Expression wird zu einem Objekt im Kontext ausgewertet wonach die Methode auf diesem Objekt aufgerufen wird.

Für die Kommunikation mit relationalen Datenbanken existieren zwei spezielle Aktivitäten. Zum einen ist dies eine SQL-Aktivität, welche

[6] http://www.jcp.org/en/jsr/detail?id=223
[7] http://uel.java.net/
[8] die hier „Felder“ bzw. „fields“ genannt werden

SQL-Skripte ausführt, zum anderen eine HQL-Aktivität[9], einer Aktivität für die Abfragesprache des ORM-Frameworks[10] Hibernate[11].

Zusätzlich enthält jPDL eine Aktivität zum Versenden von E-Mails. Wie bei den anderen technischen Service-Aktivitäten in jPDL ist auch diese Aktivität keine Beispielimplementierung der „Custom“-Aktivität, sondern ein eigenständiger Aktivitätentyp und damit ein direkter, integraler Bestandteil der Sprache jPDL.

Business-Aktivitäten jBPM besitzt kein echtes Konzept für die Realisierung und Nutzung von Business-Aktivitäten. Nichtsdestotrotz lassen sich die erwähnten „Custom“-Aktivitäten dazu nutzen, in diese Richtung zu gehen. Diese Aktivitäten werden durch Java-Klassen realisiert, welche ein bestimmtes Interface[12] implementieren müssen. Das bedeutet, dass eine Methode[13] vorhanden ist, die den Code enthält, der von der Prozess-Engine ausgeführt wird, wenn eine Prozessinstanz bei der entsprechenden Aktivität angelangt ist. Diese Methode bekommt ein Objekt[14] als Argument übergeben, welches unter anderem den Zugriff auf den Ausführungskontext erlaubt. Zur Konfiguration der Nutzung von Custom-Aktivitäten existiert die Möglichkeit, Attribute in die Java-Klasse einzufügen, welche durch Injektion direkt vor der Ausführung der Aktivität ihre Werte erhalten. Alternativ können auch Klassen-Eigenschaften[15] verwendet werden. In diesem Fall wird die Injektion über die Setter-Methode vorgenommen. Innerhalb der Implementierung der Aktivität[16] ist es möglich zu wählen, welche ausgehende Kante nach der Ausführung gewählt werden soll. Dazu ist ein entsprechender Methodenaufruf[17] im übergebenen Ausführungsobjekt durchzuführen. Die möglichen Werte werden nicht innerhalb der Aktivität definiert, sondern sind frei wählbar. Dies hat zur Folge, dass hier ein möglicher Punkt für Fehler existiert, da der Prozessmodellierer und der Aktivitätenentwickler hier exakt die gleichen Bezeichnungen wählen müssen. Alle Custom-Aktivitäten werden im Prozessmodellierungstool neben den vorhandenen Aktivitäten in der Toolbox zur Verfügung gestellt. Die Toolbox ist dabei nicht strukturiert oder kategorisiert, so dass alle Aktivitäten gleichrangig nebeneinander existieren, was bei einer größeren Anzahl schnell unübersichtlich und schließlich kaum noch benutzbar werden kann.

[9] HQL: „Hibernate Query Language“
[10] ORM: „object-relational Mapping“ (deutsch „objekt-relationale Abbildung“)
[11] http://www.hibernate.org/
[12] `ActivityBehaviour`
[13] `execute()`
[14] ein Objekt der Klasse `ActivityExecution`
[15] englisch „Properties“, also private Attribute mit Getter- und Setter-Methoden
[16] also in der Methode `execute()`
[17] `take()`

Fazit Zusammengefasst besitzt jBPM 4 eine extrem einfache Unterstützung für Business-Aktivitäten, was gleichzeitig einen Mangel an vielen wichtigen und interessanten Features bedeutet. Weder Dokumentation, noch Icons, noch die Definition ausgehender Kanten, noch Validierungsregeln werden unterstützt. Weiterhin gibt es keine Tools zur Generierung der Aktivitäten, kein Repository und die Aktivitäten werden nicht auf irgendeine Art und Weise kategorisiert. Nichtsdestotrotz können die „Custom"-Aktivitäten schnell erstellt und direkt verwendet werden. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

6.1.2 jBPM 5.x (BPMN 2.0)

Skript-Aktivitäten In Version 5 wird bei jBPM nach wie vor eine Skript-Aktivität angeboten. Der Spezifikation von BPMN 2 folgend wird diese hier „Script Task" genannt. Verschiedene Sprachen werden unterstützt, z. B. Java und MVEL[18]. Das jBPM-Benutzerhandbuch enthält dazu die folgende Warnung:

> „You should try to avoid contacting external services through a script node. (...) It is (...) interacting with external services without the knowledge of the engine, which can be problematic, especially when using persistance and transactions. In general, it is probably wiser to model communication with an external service using a service task."

Es wird also davor gewarnt, Skript-Aktivitäten zur Service-Integration zu benutzen. Andererseits tendieren auch andere Aktivitäten oft dazu wie Skript-Aktivitäten benutzt zu werden, z. B. wenn komplexe Ausdrücke als Parameterwerte verlangt werden.

Technische Service-Aktivitäten jBPM 5 setzt darauf, den Einsatz technischer Service-Aktivitäten möglichst zu vermeiden. So gibt es hier bewusst keine Aktivität wie z. B. „Call Web-Service". Stattdessen wird die Benutzung „domänenspezifischer Aktivitäten" empfohlen, welche dem Konzept der Business-Aktivitäten sehr nahe kommen.

Zwei fertig zu benutzende Aktivitäten werden bereits mit dem Produkt ausgeliefert. Dies sind eine Logging-Aktivität zur Generierung von Log-Nachrichten während der Laufzeit sowie eine Aktivität zum Verschicken von E-Mails. Diese können sowohl als technische Service-Aktivitäten als auch als Business-Aktivitäten wahrgenommen werden. Natürlich ist es

[18] http://mvel.codehaus.org/

auch immer möglich, die domänenspezifischen Aktivitäten zu nutzen um eigene technische Service-Aktivitäten zu entwickeln.

Business-Aktivitäten Im Handbuch zu jBPM 5 ist zu lesen, dass „domain specific processes“, also domänen-spezifische Prozesse unterstützt werden. Damit ist gemeint, dass Prozesse domänenspezifische Business-Aktivitäten enthalten können. In jBPM werden diese manchmal als „work items“ und manchmal als „domain specific services“ bezeichnet. Es existiert ein Komponentenmodell, welches definiert, wie eigene Aktivitäten entwickelt werden, die im Prozessmodellierungstool in der Aktivitätensammlung auftauchen und somit direkt von Modellierern benutzt werden können. Leider ist es hier genau wie bei jBPM 4 so, dass die Sammlung schnell unübersichtlich und somit schwer zu handhaben wird, da es keine Kategorien von Aktivitäten und keine Suchfunktion gibt.

Obwohl sie meist externe Services aufrufen, werden die domänenspezifischen Business-Aktivitäten nicht als Service-Tasks realisiert. Hier weicht jBPM also von der Spezifikation zu BPMN 2 ab. Stattdessen wird das sehr allgemeine Element „Task“ verwendet. Dieses wird dann durch proprietäre Erweiterungen zu einem Element, was so in BPMN 2 nicht vorgesehen ist. Ein Unterschied zum Service-Task aus dem Standard ist es hier auch erlaubt, eigene Icons für die Aktivität zu definieren, was prinzipiell zu befürworten ist. Für Service-Tasks ist in der Spezifikation vorgegeben, dass zwei überlappende Zahnräder in der oberen linken Ecke des abgerundeten Rechtecks dargestellt werden.

Eine Business-Aktivität wird definiert in einer Konfigurationsdatei („work item definition“), eine Datei mit der Endung „wid“, welche die MVEL-Syntax benutzt. Die Implementierung der Funktionalität geschieht in einer dazugehörigen Java-Klasse. Die wid-Datei ist alles, was der Modellierer benötigt. Wenn der Prozess ausgeführt werden soll, muss jede Aktivität zusätzlich eine Implementierung besitzen. Die wid-Datei definiert einen eindeutigen Namen (`name`), eine Beschriftung für die Modellierungsoberfläche (`display name`), ein Icon und die Parameter (mit ihren Datentypen).

Im Prozess-Editor können die Parameterwerte entweder mit konstanten Werten belegt werden oder durch die Definition eines Parameter-Mappings an Kontext-Variablen gebunden werden. Bei letzterer Variante werden vor der Ausführung der Aktivität die Werte vom Kontext in die Aktivität kopiert. Die Parameter werden auch für Rückgabewerte benutzt, wobei es beliebig viele Rückgabewerte geben kann. Zu dem Zweck kann ein so genanntes „Result Mapping“ definiert werden, was ein exakt spiegelbildliches Verhalten zum Parameter-Mapping besitzt. Nach der Ausführung der Aktivität werden also die Werte aus der Akti-

vität in den Kontext kopiert. Leider wird in der wid-Datei nicht zwischen Eingabe- und Ausgabeparametern unterschieden. Dies muss also aus der Dokumentation der Aktivität hervorgehen.

Zusätzlich zur hauptsächlichen Implementierung der Aktivität können auch noch Aktionen definiert werden, die bei Eintritt in eine Aktivität[19] oder bei Austritt aus einer Aktivität[20] ausgeführt werden. Diese Aktionen können für einzelne Instanzen der Aktivitäten, also für bestimmte Knoten im Prozessmodell definiert werden.

Die Java-Klasse muss wie bei der Custom-Aktivität von jBPM 4 ein bestimmtes Interface implementieren[21]. Die Methode mit dem Service-Code[22] bekommt zwei Argumente übergeben: ein Objekt, welches Zugriff auf die Parameterwerte liefert[23] und zusätzlich er `WorkItemManager`, welcher zum einen darüber informiert werden muss, wenn die Arbeit der Aktivität erledigt ist, zum anderen explizit aufgefordert werden muss, die Parameterwerte anhand des Result-Mappings in den Kontext zu kopieren.

Jede Service-Aktivität[24] hat bei jBPM grundsätzlich nur eine ausgehende Kante. Für Entscheidungen wird das XOR-Gateway von BPMN 2 benutzt.

Die Aktivitäten können in einem zentralen Repository bereitgestellt werden, auf das per Netzwerk zugegriffen werden kann. Sie werden in Kategorien eingeordnet und können in den Prozess-Editor importiert werden, wo sie in der lokalen Aktivitäten-Sammlung auftauchen.

Fazit Zusammengefasst lässt sich sagen, dass jBPM 5 besonders durch die Unterstützung seiner „domänenspezifischen Aktivitäten“ punktet. Sie besitzen ein graphisches Icon und sind wie bei jBPM 4 in das Modellierungstool integriert. Es gibt ein Repository, in welchem die Aktivitäten kategorisiert vorgehalten werden. Auf der anderen Seite existiert keine Unterstützung für Dokumentation, keine Validierungsregeln und keine Generierungstools. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

[19] „on-entry action“
[20] „on-exit action“
[21] `WorkItemHandler`
[22] `executeWorkItem`
[23] `WorkItem`
[24] egal ob Skript-Aktivität, technische Service-Aktivität oder Business-Aktivität

6.2 Activiti (BPMN 2.0)

Skript-Aktivitäten Der Script-Task von Activiti [act13] ist dem von jBPM 4 und 5 sehr ähnlich. Alle JSR-223-kompatiblen Sprachen werden unterstützt. Direkt funktionieren Groovy und JavaScript[25].

Technische Service-Aktivitäten Für SOAP-Web-Services existiert eine spezielle technische Service-Aktivität. Zusätzlich gibt es zwei direkt zu verwendende Aktivitäten, eine zum Versand von E-Mails, und eine für die Integration mit dem Enterprise Service Bus (ESB) Mule.

Business-Aktivitäten In Activiti ist keine eindeutige Einordnung der Umsetzung der Service-Tasks möglich. Sie liegt zwischen den Kategorien der technischen Service-Aktivitäten und der Business-Aktivitäten. Einerseits gibt es wiederverwendbare Software-Komponenten namens „Java delegates“, andererseits muss der Prozessmodellierer immer noch auf einem technischen Level agieren, wie es für technische Service-Aktivitäten typisch ist. Zu jedem Zeitpunkt ist ihm sehr bewusst, dass er mit Java-Klassen hantiert und nicht mit irgendeinem beliebigen Service. Es gibt nur einen „Service-Task“ in der Aktivitäten-Sammlung, nicht eine Aktivität pro Service.

Ein Service-Task kann in Activiti auf drei verschiedene Arten benutzt werden. Was der Idee der Business-Aktivität am nächsten kommt ist der Einsatz eines „Java-Delegates“. Dies ist wieder eine Java-Klasse, welche ein bestimmtes Interface implementiert[26], so dass sichergestellt ist, dass eine `execute`-Methode vorhanden ist. Diese Methode bekommt wieder ein Objekt übergeben, über das auf den Kontext zugegriffen werden kann[27]. Im Modellierungstool muss der vollständige Klassenname des Java-Delegates (incl. Package) angegeben werden. Der Prozessmodellierer muss also den Klassennamen kennen und muss ihn eingeben, anstatt einfach einen Service auszuwählen aus einer fertigen Liste in der GUI des Modellierungstools. Es kann exakt eine Rückgabevariable existieren, welche im Modellierungstool definiert werden muss. Für Eingabe-Parameter (hier „fields“ genannt) muss ein Name sowie eine Expression angegeben werden. Letztere kann auch eine konstante Zeichenkette sein. Der Übergabemechanismus der Parameterwerte ist „Call-by-value“, die Werte werden also kopiert.

Der zweite Weg, einen Service-Task zu definieren, ist die Angabe einer Expression. Diese Expression kann ein Objekt im Kontext referenzieren,

[25] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[26] `JavaDelegate`
[27] `DelegateExecution`

darauf Methoden ausführen und diesen Methoden wiederum andere Objekte aus dem Kontext als Parameter übergeben. In diesem Fall kann ein Service-Task mit einer einfachen Skript-Aktivität verglichen werden, da eine Ausdruckssprache[28] als eine einfache Skript-Sprache gesehen werden kann. Dieser Ansatz ist also ebenfalls vergleichbar mit dem Java-Task von jPDL.

Die dritte Art, einen Service-Task zu definieren ist eine Kombination aus den beiden obig beschriebenen Ansätzen. Dabei werden eine Instanz eines Java-Delegates im Kontext und eine einfache, so genannte „Delegate-Expression" benötigt. Die Delegate-Expression enthält dabei lediglich den Kontext-Schlüssel der Java-Delegate als einzige Information.

Eine Service-Aktivität[29] besitzt bei Activiti immer nur eine ausgehende Kante. Für Entscheidungen wird das in der Spezifikation von BPMN 2 definierte XOR-Gateway verwendet.

Fazit Zusammenfassend lässt sich sagen, dass Activiti keine echte Unterstützung von Business-Aktivitäten bietet. Die Java-Delegates können höchstens als Zwischending zwischen technischen Service-Aktivitäten und Business-Aktivitäten angesehen werden. Der Benutzer muss Kenntnisse in Java besitzen, sowie die Namen der benötigten Klassen und Methoden kennen. Außerdem müssen ihm noch Interna wie Parameternamen bekannt sein, da hierfür keine Tool-Unterstützung gegeben ist. Weiterhin existiert keine Unterstützung für Dokumentation, Icons, Validierungsregeln oder die Definition ausgehender Kanten. Obwohl Activiti in anderen Belangen als eine stabile und Feature-reiche Prozess-Engine angesehen werden kann, ist die Unterstützung im Bereich der Service-Integration (noch) eine klare Schwäche. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

6.3 Bonita Open Solution (BPMN 2.0)

Bonita Open Solution [Bon13] ist ein weiteres Open-Source-BPMS, welches als Prozesssprache BPMN 2.0 einsetzt. Wie bei jBPM und Activiti handelt es sich um ein Paket aus Prozessengine, Modellierungstool und zusätzlichen Hilfstools.

[28] oder „Expression-Language"
[29] egal ob Skript-Aktivität, technische Service-Aktivität oder Business-Aktivität

Die Integration von Services wird hier mit mit so genannten „Connectoren“[30] umgesetzt. Ein Connector verbindet einen Task (bzw. eine Aktivität) oder einen Prozess (bzw. einen Pool) mit einem externen Informationssystem. Man kann also für verschiedenste Situationen definieren, dass ein externer Service aufgerufen wird:

- Beim Betreten eines Tasks
- Beim Verlassen eines Tasks
- Beim Starten eines Prozesses
- Beim Verlassen eines Prozesses

Das Connector-Prinzip stimmt wieder nicht exakt mit dem überein, was in der BPMN-Spezifikation zu „Service-Tasks“ definiert wird. Das liegt wohl daran, dass Bonita als Software deutlich älter ist als BPMN. Vor BPMN wurde eine eigene, proprietäre Prozesssprache eingesetzt. Ein Connector wird vom Prozessmodellierer benutzt, indem er zunächst zu einem Task oder Pool hinzugefügt wird. Anschließend erscheint ein Wizard zur Konfiguration des Connectors. Dieser Wizard entspricht demnach von seiner Aufgabe her in etwa dem SIB-Inspektor des jABC.

Ein Connector ist in Java implementiert, so dass der vom Task auszuführende Code auch in Java definiert ist. Dies ist ähnlich wie bei SIBs, nur dass hier ein Connector an einen beliebigen Task oder Prozess auf verschiedene Arten angehängt werden kann. Ein SIB im jABC ist immer ein eigenes, spezielles Prozesselement. Die Eingabedaten für Connectoren können entweder Konstanten sein oder Expressions, bei denen auf den Kontext zugegriffen werden kann. Die Ausgabedaten werden ebenfalls wieder in den Ausführungskontext geschrieben. Es existiert bereits eine Sammlung an Connectoren und es ist auch möglich, eigene Connectoren zu erstellen. Connectoren werden in verschiedenen Kategorien angeboten. Diese Kategorien sind jedoch fest vom Hersteller vorgegeben und sind nicht hierarchisch.

Skript-Aktivitäten Es existieren fertige Connectoren für die Ausführung von Shell-Skripten sowie für die Skriptsprache Groovy.

Technische Service-Aktivitäten Die Sammlung der fertigen Connectoren enthält auch einige technische Service-Aktivitäten. Dies sind unter anderem die folgenden:

- Java: Zur Ausführung einer Java-Methode
- Web-Services: Zum Aufruf eines SOAP-Web-Services

[30] http://www.bonitasoft.com/resources/documentation/bos-56/connectivity

- Datenbankzugriff (H2, HSQL, Oracle, JDBC, Access)
- LDAP: Zum Zugriff auf einen LDAP-Verzeichnisdienst
- E-Mail: Zum Versenden von E-Mails
- Twitter: Zum Veröffentlichen eines „Tweets“ oder Versenden einer „Direct Message“ in Twitter

Business-Aktivitäten Durch die Möglichkeit, eigene Connectoren zu erstellen, ist es möglich, Business-Funktionen einzubinden. Damit ist man konzeptionell sehr nah an den Business-Aktivitäten. Man könnte theoretisch für einen speziellen Service einen Connector erstellen, auch wenn dies nicht unbedingt immer konzeptionell vom Hersteller so vorgesehen ist. Für das Erstellen eigener Connectoren gibt es den „Connector Creation Wizard“, also einen Wizard, der (ähnlich wie die SIB-Creator-GUI des jABC) ein Code-Grundgerüst generiert. Es ist dabei möglich, einen ganz neuen Connector zu erstellen oder einen bestehenden als Vorlage zu verwenden. Der generierte Stub-Code wird nach dem Wizard-Durchlauf zur Bearbeitung angezeigt. Die eigentliche Funktionalität wird also nach wie vor von Hand implementiert.

Ein Connector besitzt die folgende Struktur:

- ConnectorId: ein eindeutiger Name
- Description: ein einzeiliges Feld mit einer kurzen Dokumentation
- Category: eine von mehreren, festen, nicht hierarchischen Kategorien
- Icon: ein graphisches Icon für den Connector
- Java-Klassenname (ohne das Package)
- Java-Packagename
- Pages: die Seiten des Konfigurations-Wizards von diesem Connector
- Outputs: die Ausgabevariablen (jeweils die Felder `name` und `data type`).

Wie Business-Funktionen hier konzeptionell eingebunden werden, kann man am besten an bestehenden Connectoren sehen. So existiert ein Connector für die SAP-BAPI. Dieser ist sehr technisch und daher eher in die Kategorie der technischen Service-Aktivitäten einzuordnen. Der Grund für diese Einordnung ist, dass es hier nur genau eine Aktivität gibt, und zwar die zum Aufruf einer Funktion. Die Benutzung ähnelt also eher der Programmierung mit JCo oder dem Umgang mit Java Reflection.

Mehrere Connectoren existieren für das CRM-System Salesforce. Diese sind ebenfalls relativ technisch. Immerhin existieren hier mehrere, also speziellere Connectoren. Für jede CRUD-Operation existiert ein Connector. Man kann also Salesforce-Objekte erzeugen, auslesen, manipulieren und löschen.

Eine relativ große Sammlung von Connectoren existiert für das DMS Alfresco. Mit 13 verschiedenen Connectoren können hier Dateien hoch- und heruntergeladen werden, Ordner erstellt werden und viele andere DMS-typische Aktionen durchgeführt werden. Diese Connectoren sind schon so speziell, dass man von Business-Aktivitäten sprechen kann.

Insgesamt lässt sich also sagen, dass die Einbindung der verschiedenen Systeme sehr unterschiedlich vollzogen wurde, was wohl vor allem auch an der Komplexität der Aufgabe liegt. Ein vollständiger fachlicher Connector für die SAP-BAPI, wo für jede Funktion ein Connector existiert, würde einen immensen Implementierungsaufwand darstellen. Entsprechende Generatoren für Backend-Systeme (wie bei InBuS) sind hier nicht vorgesehen.

Fazit Sehr positiv zu bewerten ist, dass bei Bonita Open Solution dem Thema „Connectivity“[31] ein vergleichsweise hoher Stellenwert beigemessen wird. Eine große Sammlung von Connectoren ist bereits vorhanden. Diese sind auch relativ gut und ausführlich dokumentiert. Teilweise schaffen Sie es sogar durch eine Spezialisierung auf bestimmte eingeschränkte Aufgaben, als Business-Aktivitäten gelten zu können. Bei großen Backend-Systemen wie SAP stößt dieser Ansatz jedoch schnell an seine Grenzen. Hier wird lediglich eine technische Service-Aktivität angeboten. Backend-spezifische Generatoren sind nicht vorgesehen. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

6.4 AristaFlow

AristaFlow [DRRM+10, Ari13] ist eine BPM-Plattform, bei der die Unterstützung von Ad-hoc-Prozessen und die sehr explizite Darstellung des Datenflusses die Haupt-Features darstellen. Wie die vorherig behandelten Systeme ist AristaFlow Java-basiert und unterstützt die Entwicklung eigener Aktivitäten in Java. Hier wird keine offener Standard zur Prozessmodellierung wie BPMN 2 sondern eine proprietäre Prozessmodellierungssprache eingesetzt, welche vom AristaFlow-Vorgängerprodukt ADEPT [RD98] übernommen wurde.

[31] was sinngemäß dem Thema „Service-Integration“ entspricht

Skript-Aktivitäten AristaFlow bietet eine Aktivität zur Ausführung von Skripten in der Sprache „BeanShell“[32], welche sich sehr stark an der Java-Syntax orientiert.

Technische Service-Aktivitäten Es existiert eine große Sammlung technischer Service-Aktivitäten. Typischerweise bietet hier eine Aktivität eine ganze Sammlung verschiedener Varianten eines Services an:

- Die SQL-Aktivität beinhaltet Services zum Zugriff auf relationale Datenbanken mittels SQL (DB2, Derby, MS-SQL, MySQL, Oracle).
- Die GUI-Aktivität ermöglicht die Interaktion mit einem Nutzer durch eine Eingabemaske (welche automatisch generiert werden kann).
- Die EXE-Aktivität führt Kommandozeilenbefehle des jeweiligen zugrundeliegenden Betriebssystems aus.
- Die Aktivität „StaticJavaCall“ führt eine Klassenmethode[33] einer Java-Klasse aus.
- Die Aktivität „SendMail“ dient dem Versand von E-Mails.
- Die XOR-Aktivität wird für Entscheidungen und die damit verbundenen Verzweigungen im Prozessgraphen genutzt. Dazu wird ein Ausdruck ausgewertet und abhängig vom Ergebnis der Auswertung die entsprechende ausgehende Kante gewählt.
- Die Aktivität „FileIO“ bietet zwei Services, einen zum Lesen und einen zum Schreiben von Dateien.
- Es existieren Aktivitäten zur Übertragung von Daten über ein Netzwerk mittels (S)FTP oder HTTP.
- Die Aktivität „StringFormat“ kann eine Zeichenkette mittels eines gegebenen Patterns formatieren.
- Die Aktivität „AxisWS“ ruft einen SOAP-Webservice auf.
- Die XSLT-Aktivität transformiert ein XML-Dokument durch die Ausführung eines XSLT-Skripts. Diese Aktivität kann demnach also auch als Skript-Aktivität eingeordnet werden.
- Die Aktivität „ResultSet“ enthält mehrere Services zum Umgang mit einem JDBC-Result-Set[34].

[32] http://www.beanshell.org/
[33] also eine mit den Schlüsselwort `static` deklarierte Methode
[34] JDBC: „Java Database Connectivity, http://www.oracle.com/technetwork/java/javase/jdbc/index.html

Business-Aktivitäten In AristaFlow ist es prinzipiell möglich, eigene Aktivitäten zu erstellen. Diese Vorgehensweise wird jedoch nicht von Herstellerseite beworben. Das Konzept sieht hier eher die Nutzung der zahlreichen technischen Service-Aktivitäten vor. In persönlicher Kommunikation mit dem Hersteller stellt sich heraus, dass die Aktivitäten im Normalfall vom Hersteller selbst oder durch unterstützende Studenten implementiert werden. Eine Entwicklung durch Kunden selbst ist eher unüblich. Es wurde jedoch angeboten, dass man in entsprechenden Schulungen durch den Hersteller die Erstellung lernen kann. Später wurde mir dann doch ein Dokument bereitgestellt, in dem recht ausführlich dargelegt wurde, wie eigene Aktivitäten entwickelt werden können. Beim Studium dieser Unterlagen stellte sich heraus, dass die Entwicklung eigener Aktivitäten zur Service-Integration und damit die Bereitstellung von Business-Aktivitäten sehr wohl möglich ist und weder besonders schwierig noch aufwändig ist.

Wie bei anderen Prozess-Frameworks muss zur Realisierung einer Aktivität eine Java-Klasse angelegt werden, die ein gegebenes Interface[35] implementiert. Zusätzlich existiert eine abstrakte Implementierung dieser Schnittstelle[36], welche zur Verwendung empfohlen wird, da hier viele Dinge fertig implementiert sind, die man sonst immer wieder neu erstellen müsste. Der auszuführende Code muss auch hier wieder in eine bestimmte Methode des Interfaces eingefügt werden[37]. Die Oberklasse besitzt ein Feld[38], welches mit `protected` gekennzeichnet ist und dadurch auch in der Aktivitätenimplementierung benutzt werden kann. Über dieses Feld kann auf verschiedene Dinge zugegriffen werden, die während der Ausführung benötigt werden, z. B. ein `dataContext` um Eingabedaten zu lesen oder Ausgabedaten zu schreiben oder das `runtimeEnvironment` um die Prozess-Engine zu informieren, dass die Ausführung der Aktivität beendet ist.

Eine fertige Aktivität muss in eine jar-Datei gepackt werden, welche dann im „Activity Repository“ registriert werden muss. Letzteres ist eine eigenständige Server-Anwendung, die ein zentrales Repository für alle Aktivitäten bereitstellt. Das Deployment wird über den so genannten „Repository Editor“ vorgenommen, einer Client-Software für die Verwaltung des Activity Repositorys. Der Benutzer muss hier eine neue „Komponente“ anlegen und anschließend darin eine neue „Operation“. Anschließend erscheint ein Wizard, in dem unter anderem die Parameter der Aktivität definiert werden. Für jeden Parameter müssen Name, Datentyp und Datenflussrichtung (Eingabe oder Ausgabe) definiert werden. Hier

[35] hier: `ExecutableComponent`
[36] mit Namen `ExecutionEnvironment`
[37] hier: `run()`
[38] mit Namen `sessionContext`

ist es sehr wichtig, exakt die Namen, Typen und Datenflussrichtungen anzugeben, die auch in der Implementierung der Aktivität benutzt wurden. Es existiert kein unterstützender Mechanismus, der sicherstellt, dass man hier keine fehlerhaften Referenzen verwendet. Weichen die Angaben von der Implementierung ab, ist die Aktivität nicht oder nur fehlerhaft ausführbar. Es ist auch möglich, dokumentierenden Text zur Aktivität und zu den Parametern hinzuzufügen. Alle im Repository vorgehaltenen Aktivitäten können direkt im so genannten „Template-Editor“ (also dem Tool zur graphischen Modellierung der Prozessgraphen) benutzt werden. Hier ist es angenehm, dass es möglich ist, den Aktivitäten individuelle Icons zuzuweisen, die ihre Semantik symbolisieren.

Leider wird die Dokumentation der Aktivitäten nicht in der baumartigen Übersicht aller Aktivitäten im Template-Editor angezeigt. Erst wenn eine Aktivität bereits im Prozessmodell platziert wurde, ist die Dokumentation einsehbar. Die Konfiguration des Prozesses (und dessen Aktivitäten) wird durch Wizards geleitet, welche den Benutzer nacheinander durch alle benötigten Schritte leitet.

Eine Service-Aktivität hat in AristaFlow grundsätzlich nur eine ausgehende Kante. Für Entscheidungen wird die spezielle XOR-Aktivität benutzt.

Fazit Zusammenfassend lässt sich sagen, dass AristaFlow mit der ausgereiften Unterstützung der Entwicklung eigener Aktivitäten punktet und dies obwohl dieser Aspekt eigentlich nicht dem primären Konzept von AristaFlow entspricht. Besonders das Repository mit dem Repository-Editor sind hier positiv hervorzuheben. Die Aktivitäten besitzen ihr eigenes Icon, es existiert Unterstützung für Dokumentation und es gibt sogar einfache Generierungstools (z. B. der direkte Import von Java-Methoden oder SOAP-Web-Services in den Repository-Editor).

Auf der anderen Seite könnte der Prozess-Editor noch dahingehend verbessert werden, dass die Dokumentation der Aktivitäten auch in der Aktivitäten-Übersicht einsehbar ist. Außerdem gibt es keine Unterstützung für beschriftete, ausgehende Kanten oder Validierungscode. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

6.5 YAWL

„YAWL“ [The13c, HAAR10] steht für „Yet Another Workflow Language“ und bezeichnet sowohl eine Petri-Netz-basierte Prozessbeschreibungssprache als auch das dazugehörige BPMS, welches als Open-Source-

Software frei verfügbar ist. Die Sprache YAWL wurde mit dem Ziel entwickelt, eine Prozesssprache zu besitzen, die alle so genannten „Workflow-Patterns“ [DATHKB03] direkt unterstützt. Workflow-Patterns sind dabei typische Kontrollfluss- und Datenflusselemente, die als strukturierende Elemente in verschiedenen Prozesssprachen vorkommen. Beispiele sind die Sequenz[39], die parallele Verzweigung[40] oder die exklusive Auswahl[41]. Ein Workflow-Pattern entspricht also in etwa der Einführung eines neues Control-SIBs im jABC. Insgesamt werden aktuell auf der Homepage [Wor13] 43 verschiedene Patterns gelistet. Daraus folgt, dass es sich bei YAWL um eine sehr komplexe Sprache handelt, da für jedes Pattern ein entsprechendes Element vorhanden sein muss, welches von Modellierer verstanden und richtig eingesetzt werden muss.

Nicht nur die Sprache selbst ist sehr komplex. Gleiches gilt ebenso für den Mechanismus zur Service-Integration, welcher sich stark von den Konzepten der oben beschriebenen BPMS abhebt. Für die Einbindung von Services werden so genannte „Custom YAWL-Services“ [The13b, The13a] eingesetzt. Dies sind mit SIBs vergleichbare Prozesskomponenten, nur dass diese nicht von der Prozessengine selbst sondern entfernt ausgeführt werden. Die Services werden also auf einem dedizierten Server bereitgestellt und dann von der Prozessengine bei Bedarf über das Netzwerk angestoßen. Die Kommunikation erfolgt dabei mittels XML-Dokumenten, die über HTTP verschickt werden. Das Konzept entspricht dem der „Worklets“, die von Michael Adams in Kapitel 4 seiner Dissertation [Ada07] beschrieben werden. Durch die Übertragung von XML über HTTP wird eine Unabhängigkeit sowohl von der eingesetzten Plattform als auch von der Programmiersprache erreicht. Ein YAWL-Service kann also prinzipiell in jeder beliebigen Programmiersprache implementiert werden. Dies ist ein wesentlicher Unterschied zu den vorher untersuchten BPMS. Einzige Voraussetzung ist, dass der YAWL-Service auf HTTP-Anfragen reagieren und mit XML umgehen kann. Beides sollte mit jeder relevanten Programmiersprache möglich sein. Es existiert bereits eine fertige Server-seitige Implementierung, die als Grundlage für die Entwicklung von YAWL-Services benutzt werden kann, und zwar eine Java-Servlet-Implementierung. Diese ist lauffähig auf allen gängigen Servlet-Engines wie Tomcat[42] oder Jetty[43].

Ein YAWL-Service hat vier Aufgaben:

1. Empfangen der Nachricht von der Engine, dass der dazugehörige Task erreicht wurde, der Service also ausgeführt werden muss

[39] „Sequence“
[40] „Parallel Split“
[41] „Exklusive Choice“
[42] http://tomcat.apache.org/
[43] http://www.eclipse.org/jetty/

2. Die Engine informieren, dass der Service die Verantwortung für die Ausführung übernommen hat
3. Den Service ausführen
4. Die Engine informieren, dass der Service ausgeführt wurde. Damit wird der Engine erlaubt, die Ausführung des Prozesses fortzusetzen.

Skript-Aktivitäten Skript-Aktivitäten sind bei YAWL nicht vorgesehen.

Technische Service-Aktivitäten Es existieren mehrere fertige Implementierungen von YAWL-Services. Zum Aufruf von SOAP-Web-Services existiert der „YAWL WS-Invoker“, welcher die Technologie Apache WSIF[44] einsetzt. Weitere Services versenden SMS, E-Mails oder Nachrichten über den Dienst Twitter. Außerdem gibt es noch einen YAWL-Service zum Umgang mit digitalen Signaturen.

Business-Aktivitäten Durch die Möglichkeit, eigene YAWL-Services zu implementieren, ist es auch möglich, so Business-Aktivitäten zu erstellen. Technische Unterstützung, z. B. durch Code-Generatoren, gibt es nicht.

Fazit Das Service-Integration-Konzept von YAWL besitzt den besonderen Pluspunkt der absoluten Unabhängigkeit von Plattform und Programmiersprache. Kritisch anzumerken ist dabei, dass es auch nachteilig sein kann, jeden einzelnen Service, auch sehr kleinschrittige, über das Netzwerk aufzurufen. Hier sollte man sehr darauf achten, nur grobgranulare Prozesse zu modellieren, da ansonsten der Overhead für die Nachrichtenübertragung zu groß wird. Business-Aktivitäten sind mit YAWL zwar möglich, werden jedoch nicht durch entsprechende Software unterstützt. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

6.6 Windows Workflow Foundation

Die Microsoft-Software „Windows Workflow Foundation“ [Col10] (oft auch als „WF“[45] abgekürzt) wird selbst nicht als BPMS vermarktet,

[44] WSIF: Web Services Invocation Framework
[45] nicht „WWF“!

besitzt jedoch viele Eigenschaften eines BPMS, so dass es gut vergleichbar mit den anderen untersuchten BPMS ist. WF ist teil der .NET-Bibliothek und wir auch entsprechend von der Entwicklungsumgebung Visual Studio unterstützt. Es handelt sich im Wesentlichen um eine allgemeine Bibliothek zur Umsetzung ausführbarer Workflows. Workflows können in Visual Studio graphisch modelliert werden und anschließend direkt ausgeführt werden. WF ist damit also gut vergleichbar mit Bibliotheken wie jBPM, Activiti oder jABC. Obwohl es selbst nicht als BPMS beworben wird, ist es jedoch technische Grundlage anderer prozessorientierter Microsoft-Produkte, wie dem ESB „BizTalk“ oder der Unternehmens-Webanwendung „Sharepoint“.

Ein WF-Prozess besteht aus einzelnen Aktivitäten, die hier auch „Activity“ genannt werden. Ein Prozess selbst ist dabei wieder eine Activity. Somit existiert ein hierarchisches Prozessmodell wie beim jABC. Bei der Prozessmodellierung wird grundsätzlich direkt im Hintergrund entsprechender Code generiert. Hier werden zwei verschiedene Sprachen eingesetzt, wobei optional eingestellt werden kann, ob nur eine von beiden Sprachen oder beide generiert werden. Zum einen gibt es hier die XML-basierte Beschreibungssprache XAML[46], zum anderen C#. Soll der Prozess ausführbar sein, so muss in jedem Fall C# generiert werden. Obwohl direkt Code generiert wird, ist es trotzdem möglich, mit einem graphischen Debugger (vergleichbar mit der Tracer-GUI des jABC) durch den Prozess zu navigieren. Entsprechend notwendige Voraussetzungen dafür werden direkt mit in den Code hinein generiert.

Jede einzelne Activity in einem Prozess ist wieder in C# implementiert. Es ist hier also auch möglich, eigene Aktivitäten zu implementieren. Zur Kommunikation mit externen Services wird hauptsächlich auf WCF[47] zurückgegriffen, neben WF einem weiteren integralen Bestandteil von .NET, in diesem Fall zur Kommunikation über Web-Services.

Skript-Aktivitäten In WF gibt es eine so genannte „Code-Activity“[48]. Damit kann C#-Code direkt in den Workflow eingebettet werden. Somit entspricht dies einer Skript-Aktivität.

Technische Service-Aktivitäten Es existiert zwar eine Sammlung von fertigen Activities[49], davon sind jedoch die meisten Kontrollflussaktivitäten, vergleichbar mit den Control-SIBs des jABC, also zuständig für

[46]XAML: Extensible Application Markup Language
[47]WCF: Windows Communication Foundation
[48]http://msdn.microsoft.com/en-us/library/vstudio/system.workflow.activities.codeactivity(v=vs.90).aspx
[49]http://msdn.microsoft.com/en-us/library/vstudio/ms733615(v=vs.90).aspx

Schleifen, Verzweigungen, Parallelität und ähnliches. Zur Integration von Services existieren hier lediglich zwei Aktivitäten:

- CallExternalMethodActivity: zum Aufruf eines lokalen Services (also einer C#-Methode)
- InvokeWebServiceActivity: zum Aufruf von WSDL-SOAP-Web-Services über WCF

Business-Aktivitäten Da es möglich ist, eigene Aktivitäten zu implementieren[50], ist es auch möglich, Business-Aktivitäten zu entwickeln. Untersützung durch Generatoren oder fertige Implementierungen von Business-Aktivitäten gibt es jedoch nicht. Die konkrete Integration von Services, besonders von Business-APIs, geht wohl über den Aufgabenbereich von WF hinaus und wird eher darauf aufbauenden Lösungen wie BizTalk zugeordnet.

Um eine Aktivität zu implementieren, muss die Klasse `Activity` erweitert werden. Eine Beispielimplementierung ist in Listing 6.1 zu sehen. Hier ist zu erkennen, dass der auszuführende Code in der dafür vorgesehenen Methode `Execute` platziert werden muss, welche den Ausführungskontext übergeben bekommt.

Listing 6.1: Beispielimplementierung der Execute-Methode

```
1 protected override ActivityExecutionStatus Execute
2     (ActivityExecutionContext executionContext)
3 {
4     ...
5     return ActivityExecutionStatus.Closed;
6 }
```

Im Unterschied zu den meisten anderen Prozessengines wird bei WF für die Kommunikation zwischen den Aktivitäten kein Variablen-Kontext eingesetzt. Der übergebene `ActivityExecutionContext` stellt keinen Daten-Container dar, sondern lediglich eine Sammlung einfacher Funktionen, mit denen die Ausführung gesteuert werden kann. Zur Kommunikation zwischen den Aktivitäten werden stattdessen so genannte „Dependency Properties“ eingesetzt. Jede Aktivität kann diese optional enthalten. Da auch der Prozess selbst wieder eine Aktivität ist, besitzt auch dieser potentiell Dependency Properties. Nun ist es möglich, zwischen den Properties (z. B. von einer Aktivität und dem umgebenden Prozess) Bindings zu definieren. So wird explizit definiert, wie die Aktivitäten miteinander kommunizieren können. Dependency Properties sind dabei relativ kompliziert zu definieren. Der beschriebene Vorgang über

[50] http://msdn.microsoft.com/en-us/magazine/cc163504.aspx

Properties und Bindings zwischen den Properties ist recht komplex und technisch. Dies ist auch Microsoft bewusst. Sie verweisen dabei auf die gute Unterstützung durch die Entwicklungsumgebung Visual Studio mit dessen so genannten „code snippet support“, der „auf Knopfdruck“ gewünschte Code-Segmente einfügt bzw. vervollständigt. Somit wird die Erstellung zwar vereinfacht, die Lesbarkeit und Wartung bleibt jedoch schwierig.

Fazit Windows Workflow Foundation ist ein Prozess-Framework, bei dem Service-Integration keine große Rolle spielt. Dies wird bei Microsoft eher darauf aufbauender Software wie BizTalk überlassen. Windows Workflow Foundation besitzt jedoch mit der Möglichkeit der Entwicklung eigener Aktivitäten das Potential zur Realisierung von Business-Aktivitäten. Direkte Unterstützung durch Generierungstools oder fertige Business-Aktivitäten gibt es nicht. Eine strukturierte Zusammenfassung der Evaluationsergebnisse und ein Vergleich mit den anderen BPMS ist in den Tabellen 6.1 und 6.2 dargestellt.

6.7 BPEL

BPEL [OAS07] steht für „Business Process Execution Language“[51] und war die erste weit verbreitete Definitionssprache für ausführbare Geschäftsprozesse. Dabei werden in BPEL die Prozesse als Orchestrierungen von Web-Services in einer XML-basierten Syntax verfasst. Die Spezifikation beinhaltet keine Vorgaben für eine entsprechende graphische Notation. BPEL ist demnach durch den Fokus auf XML und Web-Services sehr technisch und kann nicht direkt von fachlichen Prozessmodellierern benutzt werden, die keine IT-Fachleute sind.

BPEL stellte einen großen Hype dar, der mittlerweile abgeklungen ist. Heutzutage wird BPEL stark von BPMN 2.0 verdrängt. Dieser Trend ist auch in Abbildung 6.1 zu erkennen. Da BPEL selbst keine graphische Notation besaß, wurde lange versucht, BPEL aus verschiedenen anderen Sprachen (z. B. BPMN 1.x oder EPKs) zu generieren. Da diese Sprachen jedoch meist grundlegend unterschiedlich sind, war dies immer sehr problematisch und führte nie zu zufriedenstellenden Lösungen. Zum einen

[51] Der vollständige und korrekte Name der Sprache lautet seit der aktuellen Version 2.0 „WS-BPEL“, wobei das „WS“ für Web-Services steht und zeigt, dass sich diese Sprache in die lange Liste der existierenden „WS-*“-Sprachen einreiht, die verschiedene Aspekte des Umgangs mit SOAP-basierten Web-Services abdecken. Vorher hieß die vollständige Bezeichnung „BPEL4WS“. Im allgemeinen Sprach- und Schriftgebrauch hat sich jedoch die einfache Kurzform „BPEL“ durchgesetzt.

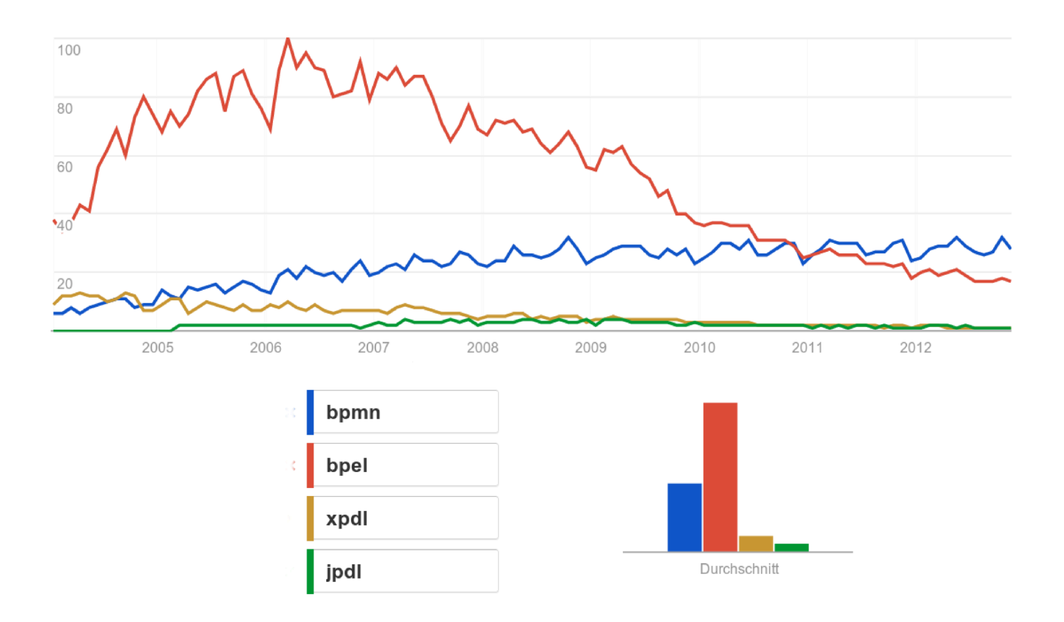

Abbildung 6.1: Relative Häufigkeiten der Erwähnungen von Prozesssprachen im WWW nach „Google Trends“ (http://www.google.de/trends/) vom 13.02.2013

sind die meisten graphischen Notationen Graph-basiert, BPEL dagegen Block-orientiert, zum anderen enthält ein BPEL-Dokument deutlich mehr technische Informationen als jedes EPK oder BPMN-Diagramm.

Bei BPEL handelt es sich zwar um eine Sprache und nicht um ein konkretes BPMS, es ist jedoch trotzdem sinnvoll, sie als Vergleich zu den anderen BPMS heran zu ziehen. Zum einen ähneln sich verschiedene BPEL-basierte BPMS stark (besonders in Bezug auf das Thema „Service-Integration“), zum anderen war BPEL lange der de facto Standard in Bereich BPM, so dass eine Nichtbetrachtung hier nicht in Frage kommt. Da BPEL sich beim Thema „Service-Integration“ im Wesentlichen auf die <invoke>-Aktivität beschränkt, wurde die Sprache in den Tabellen 6.1 und 6.2, welche einer Übersicht der Auswertung enthalten, ausgelassen.

In mehreren Diplomarbeiten wurde am Lehrstuhl 5 für verschiedene Anwendungsbereiche evaluiert, wie gut sich BPEL für die Definition von ausführbaren Geschäftsprozessen einsetzen lässt.

Jörn Gaeb verfasste eine Arbeit über die Modellierung von BPEL-Prozessen im jABC [Gae07]. Dazu implementierte er ein entsprechendes jABC-Plugin. Mit diesem war es möglich, BPEL-Prozesse in das jABC zu importieren, sie dort als SLG zu bearbeiten und diese auch wieder zu exportieren. Damit dies möglich ist, wurde zusätzlich eine spezielle Palette von BPEL-SIBs entworfen. Dabei handelt es sich um Control-SIBs,

die jeweils eine BPEL-Aktivität darstellen[52]. Hier wurde besonders deutlich, das ein SLG und ein BPEL-Prozess von grundsätzlich verschiedener Natur sind. Ein SLG ist Graph-orientiert, ein BPEL-Prozess prinzipiell Block-orientiert. Das bedeutet, dass in BPEL z. B. für eine Verzweigung mit „if“ ein entsprechender Block mit Anfang und Ende existiert. Im jABC wurde also jeweils ein SIB für den Anfang und für das Ende des if-Blocks bereit gestellt. Bei anderen Aktivitäten wurde ähnlich verfahren. Dies resultierte in einer sehr unintuitiven und komplizierten Modellierung der SLGs. Außerdem war es hier nicht möglich, beliebige SIBs zu benutzen. Stattdessen war die Modellierung auf die BPEL-SIBs beschränkt. Als Resümee lässt sich sagen, dass dieser Ansatz, eine Brücke zwischen Modellierung und ausführbaren Prozessen nicht den erwünschten Erfolg gebracht hat.

In einer zweiten Diplomarbeit wurde ein anderer Weg evaluiert. Norman Braß beschäftigte sich mit der Nutzung von BPEL für durch RFID-Technologie unterstützte Logistikprozesse. Die gewählte Software war dabei der WebSphere Process Server[53], eine BPEL-Engine von IBM. Zu der Engine liefert IBM auch die entsprechenden Modellierungstools, unter anderen den „WebSphere Business Modeler“ und den „WebShere Integration Developer“. Der Business Modeler ermöglicht die graphische Modellierung der fachlichen Geschäftsprozesse. Das Tool ist dabei für den Anwendungsexperten (also für einen Nicht-IT-Fachmann) gedacht. Das Modell ist dabei graph-orientiert. Aus diesem Modell wird anschließend BPEL-Code generiert. Der Integration Developer ist dann der Editor für den BPEL-Prozess. Hier können nun technische Details hinzugefügt werden, bis der Prozess wirklich ausführbar ist. Zum Schluss kann der Prozess auf den Process Server deployt werden. Diese Herangehensweise funktioniert schon etwas besser als die in der Arbeit von Jörn Gaeb. Trotzdem gibt es hier erhebliche konzeptionelle Probleme. Wenn aus dem Graph-orientierten fachlichen Modell BPEL generiert werden sollte, so wurde grundsätzlich alles in eine Flow-Umgebung gewandelt, also ein Block in BPEL, innerhalb dessen wieder graph-orientiert modelliert wird. Hier werden prinzipiell alle Unterelemente parallel ausgeführt. Zusätzlich ist es möglich, so genannte „Links“ einzuführen. Dies sind Contraints, die aussagen, dass eine bestimmte Aktivität erst dann ausgeführt werden darf, wenn eine bestimmte andere erfolgreich abgeschlossen wurde. Wurde nun also eine einfache Sequenz aus fünf Schritten modelliert, so resultierte dies in einem BPEL-Prozess mit einer Flow-Umgebung, die fünf parallele Threads enthält, wobei jeder der Threads jeweils darauf wartet, dass die Ausführung seines Vorgängers abgeschlossen wurde. Bei langen Sequenzen kann dies schnell zu einem Performance-Problem wer-

[52]z. B. Invoke, Receive, Reply, If oder While
[53]http://www-03.ibm.com/software/products/de/de/wps/

den. Zur Zeit der Diplomarbeit war es sogar so, dass die aktuelle Implementierung des Process Servers nicht mit sehr vielen parallelen Threads umgehen konnte, was dazu führte, dass eine konsequente Benutzung der IBM-Tools schon bei überschaubaren Prozessen zu regelmäßigen Server-Crashs führte.

Der Ansatz der BPEL-Generierung von IBM ist bei weitem nicht der einzige Versuch, auf diese Weise die Brücke von der Business-Seite zur IT zu schlagen. Auch in der BPMN-Spezifikation [OMG11] gibt es ein eigenes Kapitel, welches jedoch nur grundsätzliche Parallelen der beiden Sprachen bzgl. einzelner Elemente aufzeigt und keine detaillierte Anleitung liefert. Das grundsätzlich zu lösende Problem ist in allen Arbeiten das gleiche, und zwar die Transformation eines beliebigen Graphen in eine strukturierte, Block-orientierte Sprache [KMWL09]. Ein große Anzahl wissenschaftlicher Arbeiten verfolgen dabei grundsätzlich den gleichen Ansatz, die Graphstruktur zu analysieren und dabei die strukturierten Elemente zu identifizieren [Whi05, ODTHA06, Gao06, ODBH06, YZZ+07, ODHVDA08, SKLN09, GX09, DJB+09, PGBW09, MH11, DGBP11]. Diese Strukturen werden dann auf die entsprechenden Elemente der Block-orientierten Sprache übertragen. Einfach strukturierte Prozesse können so komplett und ohne große Transformation übernommen werden. Existieren jedoch Elemente, die von dieser Struktur abweichen, wie zum Beispiel Sprünge aus eine Schleife heraus, so müssen alternative Wege gefunden werden. Hier können entweder Teile des Prozesses dupliziert werden, was zu einer Explosion des Modells führen kann oder es wird für solche Zwecke auf eine Art von Ereignisbehandlung gesetzt. Man löst sich also an dieser Stelle von der Modellierung durch Prozesse und definiert die Abläufe stattdessen sehr lose durch das Auslösen von Ereignissen an der einen Stelle und das Reagieren auf selbige an einer anderen. Diese Ereignisse können bei BPEL zum Beispiel Web-Service-Aufrufe sein. Somit hat man auch hier wieder eine GOTO-Semantik. In jedem Fall unterscheidet sich der BPEL-Prozess stark vom ursprünglich modellierten BPMN-Prozess. Wird nun der BPEL-Prozess ausgeführt und beobachtet, so ist es schwierig, Rückschlüsse auf das Originalmodell zu ziehen. Der BPM-Kreislauf (siehe Abbildung 2.1 in Abschnitt 6.7) wird also durchbrochen. Auch in [RM06] und [WDGW08] wird das Problem des konzeptionellen Unterschieds der beiden Sprachen beschrieben. So lässt sich schließlich schlussfolgern, dass eine direkte Ausführung der Modellierungssprache (wie bei BPMN2 oder jABC) die zu bevorzugende Lösung darstellt.

Pjotr Kaspczak nutzte in seiner Diplomabeit BPEL zur Implementierung einer serviceorientierten Architektur für ein so genanntes „Campus-Management-System“, also einer Verwaltungssoftware für Universitäten [Kas09]. Eingesetzt wurde hier die BPEL-Software „Glassfish-ESB“ von

Sun Microsystems, was heute zu Oracle gehört. Glassfish-ESB ist ein Paket aus einem BPEL-Modellierungstool auf Basis der IDE Netbeans, dem JavaEE-Server Glassfish und der BPEL-Engine OpenESB. Hier wurde BPEL direkt im BPEL-Modellierungstool erstellt, also ohne zusätzliches fachliches Modell. Wichtige Erkenntnisse dieser Arbeit waren, dass die direkte Nutzung von BPEL extrem technisch ist und nur Programmierern zugemutet werden kann. Es kam häufig vor, dass der graphische Modellierer nicht ausreichte und BPEL im XML-Quelltext bearbeitet werden musste. Speziell bei Datentransformationen wurde es kompliziert. Für einfache Transformationen reichte hier der „Mapping Editor“, bei dem per Drag and Drop Pfeile zwischen den Blättern zweier Bäume gezogen werden. Bei komplexeren Transformationen wurde auf XSLT gesetzt. Die Ausführung von XSLT ist jedoch kein Bestandteil der offiziellen BPEL-Spezifikation. Stattdessen wurde hier eine proprietäre Erweiterung von Sun Microsystems eingesetzt. Hier zeigt sich ein weiteres Problem, was oft bei der Nutzung von „Standards“ anzutreffen ist. Die einzelnen Hersteller halten sich so gut wie nie an den Standard, sondern implementieren nur den Teil, der ihnen zusagt und erweitern dies dann noch um eigene Elemente. So ist es nicht möglich, den hier erstellten BPEL-Prozess direkt auf eine BPEL-Engine eines anderen Herstellers als Sun Microsystems zu deployen.

Skript-Aktivitäten Als Skript-Aktivität in BPEL kann die Erweiterung „BPELJ“ aufgefasst werden. Hier wird Java-Code direkt in das XML von BPEL eingefügt, so dass Java und BPEL in einer Datei gemischt werden. Diese Erweiterung existiert jedoch lediglich als Spezifikation. Kein Hersteller hat dieses Feature in sein Tool integriert.

Technische Service-Aktivitäten In BPEL wird zum Aufruf von Services die Aktivität <invoke /> eingesetzt. Diese ruft SOAP-basierte Web-Services auf. Für die Benutzung dieser Aktivität sind auf Seiten des Modellierers Kenntnisse aller beteiligten Technologien notwendig, also XML, XML-Schema, WSDL, evtl. XSLT und bei Bedarf verschiedene andere WS-*-Sprachen wie WS-Security oder WS-Policy. Die Invoke-Aktivität ist also eine technische Service-Aktivität.

Business-Aktivitäten Business-Aktivitäten gibt es nicht direkt als Bestandteil der Sprache. BPEL selbst ist immer sehr technisch. Außerdem ist die Menge der Aktivitäten fest in der Spezifikation der Sprache vorgegeben. Erst durch eine gute Integration mit zum Beispiel einer ESB-Software, könnte eine gute Integration möglich sein. Dabei müsste

das BPEL-Modellierungstool den eigentlichen BPEL-Code gut verstecken und eine eigene Business-Sicht darüber aufbauen. Wie schon oben bei IBM beschrieben, ist dies jedoch keine einfache Aufgabe. Es ist auch kein BPEL-Tool bekannt, welches diese Aufgabe wirklich erfolgreich gemeistert hat.

Fazit Die Ergebnisse der Diplomarbeiten sowie die Betrachtung des Service-Integrations-Aspektes führen zu der Erkenntnis, dass BPEL keine gute Wahl zur Definition ausführbarer Geschäftsprozesse darstellt, besonders dann nicht, wenn Einfachheit und ein gutes „Business-IT-Alignment" gefordert sind. Zu Recht ist der BPEL-Hype abgeebbt und durch die Herangehensweise von BPMN 2.0 ersetzt worden, das fachliche Modell direkt auszuführen. Genau dies ist auch bei XMDD zentraler Bestandteil des Konzeptes.

6.8 Vergleich

Alle in den Abschnitten 6.1 bis 6.6 untersuchten BPMS unterstützen wie das in Kapitel 5 behandelte jABC die Erstellung eigener Prozesskomponenten und damit die Entwicklung von domänenspezifischen Business-Aktivitäten. Nur die Sprache BPEL (siehe Abschnitt 6.7) und damit die entsprechenden BPMS, die diese Sprache einsetzen beschränkt sich bei der Service-Integration darauf, eine einzige technische Aktivität anzubieten, in diesem Fall die Aktivität <invoke>. Betrachtet man bei den verschiedenen BPMS jedoch genauer, wie komfortabel die Unterstützung von domänenspezifischen Business-Aktivitäten ist, so wird deutlich, dass dies jeweils sehr unterschiedlich sein kann. So bietet z. B. Activiti (siehe Abschnitt 6.2) nur eine sehr rudimentäre Unterstützung an, so dass immer noch ein gewisses Maß an technischem Know-How notwendig ist. Hier wird nicht vollständig von der eingesetzten Programmiersprache Java abstrahiert. Andere Systeme wie jBPM 5 (siehe Abschnitt 6.1.2, AristaFlow (siehe Abschnitt 6.4 oder Bonita Open Solution (siehe Abschnitt 6.3) sind da schon deutlich weiter und ermöglichen die komfortable Erstellung eigener Komponenten, die Bereitstellung in Repositories sowie im Falle von AristaFlow und Bonita auch einfache Mechanismen zur Generierung der Komponenten.

Betrachtet man die verschiedenen Herangehensweisen aus einem technischen Standpunkt, so fällt auf, dass zunächst die Definition sowie die Implementierung der Aktivitäten mit unterschiedlichen Mitteln erfolgen kann. Die Implementierung erfolgt hier fast immer in Java, Windows Workflow Foundation (siehe Abschnitt 6.6) setzt hier als Teil von .NET

natürlich auf C#. Die Definition kann entweder auch in der Programmiersprache erfolgen (jABC, jBPM4, Bonita, YAWL[54]) oder in einer separaten Definitionsdatei (wie die WID in jBPM5) oder durch die Benutzung der Repository-Management-GUI (wie bei AristaFlow) oder im Prozess-Editor (wie bei Activiti). Letzteres hat den großen Nachteil, dass die Definition nicht wiederverwendet werden kann. Sie muss bei jeder Nutzung der Aktivität von neuem angegeben werden. Die AristaFlow-Lösung ist hier schon etwas besser zu bewerten, da hier die Definition nur einmal angegeben werden muss. Es gibt jedoch auch hier keine Hilfe bei der Vermeidung von Fehlern, wie z. B. Tippfehlern in Parameternamen. Das gleiche gilt für die Benutzung einer Definitionsdatei wie in jBPM 5. Bei den restlichen Lösungen, die die Programmiersprache selbst zur Definition einsetzen, kann die verwendete IDE durch fortschrittliche Syntaxchecks dabei helfen, solche Fehler zu vermeiden.

Ein weiterer Punkt ist die Behandlung ausgehender Kanten. Im jABC wird im SIB selbst definiert, welche möglichen Resultate der Ausführung existieren und damit, welche ausgehenden Kanten erlaubt sind. Dies ist eine sehr natürliche Denkweise, da sehr oft eine bestimmte Aktion verschiedene Resultate liefern kann, die jeweils die passende Nachfolgeaktion zur Behandlung dieser Resultate erfordert. Die BPMN-Tools wie Activiti, jBPM 5 oder Bonita erlauben hier immer nur eine ausgehende Kante. Für Verzweigungen werden die entsprechenden BPMN-Gateways eingesetzt, was insgesamt umständlicher, aufwändiger und weniger natürlich ist. Das selbe gilt für die XOR-Aktivität in AristaFlow oder die Aktivität für das Exclusive-Choice-Pattern in YAWL.

Um dem Business-Experten einen intuitiven Zugang zu den vorhandenen Aktivitäten zu bieten, ist eine angemessene Dokumentation ein Muss. Diese sollte Teil der Aktivitätsdefinition sein und auch direkt im Prozess-Editor angezeigt werden, wie dies im jABC der Fall ist. Ein illustrierendes Icon für die Business-Aktivität (wie bei jBPM 5, AristaFlow, WF und jABC) kann die Verständlichkeit des Prozessmodells weiter verbessern.

Die Generierung von Service-Aktivitäten (sei es voll- oder halbautomatisch) ist ein wichtiges Thema und wird von AristaFlow und Bonita teilweise unterstützt, vom jABC sehr systematisch und vielfältig. Dies kann dem Business-Experten dabei helfen, schnell und einfach neue Services zu integrieren. Der grundsätzliche Ansatz der Generierung von Business-Aktivitäten wie es Kapitel 5 beschrieben wird, ist jedoch nicht beschränkt auf das jABC, sondern lässt sich auch auf andere BPMS übertragen. Dies ist in Kapitel 7 genauer beschrieben.

Schließlich gewinnt die Organisation der Business-Aktivitäten bei einer wachsenden Service-Anzahl immer mehr an Bedeutung. Während

[54]siehe Abschnitt 6.5, WF

jABC die Aktivitäten in hierarchischen Kategorien („Taxonomien“) organisiert, bieten jBPM4, jBPM5 und AristaFlow lediglich einfache Kategorien. Activiti, YAWL und WF bieten gar keine Unterstützung für die Organisation der Aktivitäten an. In diesen Tools muss jeweils einfach der Name der gewünschten Aktivität bekannt sein und direkt eingegeben werden.

Insgesamt ist zu beobachten, dass viele Tools mittlerweile den Quasi-Standard BPMN2 einsetzen. Obwohl dieser viele Freiräume im Bereich der Service-Integration lässt, ist es trotzdem so, dass die einzelnen Lösungen wie jBPM 5, Activiti und Bonita sich nicht an die Spezifikation halten. So definiert Activiti nicht Web-Services sondern Java-Services als Standardtechnologie, jBPM benutzt die Service-Aktivität gar nicht und Bonita erlaubt das beliebige Anhängen von Aktionen an den ganzen Prozess, vor eine Aktivität oder nach einer Aktivität. Insgesamt sind die BPMN-Umsetzung also sehr unterschiedlich, so dass erkennbar ist, dass BPMN für den Bereich der Service-Integration kein ausreichender Standard ist.

In den Tabellen 6.1 und 6.2 ist der Vergleich der BPMS noch einmal in strukturierter und übersichtlicher Weise dargestellt.

Tabelle 6.1: BPMS-Vergleich - Teil 1 von 2

		jBPM 4	jBPM 5	Activiti	Bonita Open Solution
Skript-Aktivitäten		✓ (jUEL, JSR-233)	✓ (Java, MVEL)	✓ (Groovy, JavaScript, JSR-233)	✓ (Groovy, Shell-Skripte)
Technische Aktivitäten	Java	✓	–	✓	✓
	Web-Services	–	–	✓	✓
	E-Mail	✓	✓	✓	✓
	Logging	–	✓	–	–
	Datenbankzugriff	✓	–	–	✓
	andere	–	–	eine: ESB-Integration (Mule)	viele[a]
Business-Aktivitäten (BA)	Ort der Definition	Java-Code	Konfigurationsdatei (WID)	Prozess-Editor	Java-Code
	Parameter	✓	✓	✓	✓
	Datenfluss deklariert (Eingabe/Ausgabe)	–	–	–	✓
	Aktivitäten-Dokumentation (AD)	–	–	–	✓ (kurz)
	AD im Modellierungstool verfügbar	–	–	–	✓
	Graphisches Icon	–	✓	–	✓
	Ausgehende Kanten (AK)	✓	–	–	–
	AK in Aktivität definiert	–	–[b]	–[b]	–[b]
	Benutzungsregeln / Check-Code	–	–	–	–
Generierung (von BA)	Stub-Generierung	–	–	–	Connector-Wizard
	vollst. Generierung: ZT[c]	–	–	–	–
	vollst. Generierung: ZS[d]	–	–	–	–
Organisation (von BA)	Kategorien	–	✓/–[e]	–	✓
	hierarchische Kategorien	–	–	–	–
	lokales Repository	✓	✓	–	✓
	remote Repository	–	✓	–	–
Nutzung (von BA)	Gesamtbewertung	○[f]	○[f]	–[g]	+[h]

[a] LDAP, Twitter, SAP-BAPI, Salesforce, Alfresco
[b] stattdessen BPMN-Gateway
[c] zielgerichtet auf Technologie
[d] zielgerichtet auf System
[e] im Repository ja, im Editor nicht
[f] keine Dokumentation
[g] keine Dokumentation und Java-Kenntnisse erforderlich
[h] Dokumentation, Icon, Datenfluss, Kategorien

Tabelle 6.2: BPMS-Vergleich - Teil 2 von 2

		AristaFlow	YAWL	WF	jABC
Skript-Aktivitäten		✓ (BeanShell)	–	✓ (C#)	✓ (Groovy, BeanShell)
Technische Aktivitäten	Java	✓	–	–[b]	✓
	Web-Services	✓	✓	✓	–[a]
	E-Mail	✓	✓	–	✓
	Logging	–	–	–	✓
	Datenbankzugriff	✓	–	–	✓
	andere	viele[b]	mehrere[b]	–	viele[c]
Business-Aktivitäten (BA)	Ort der Definition	Repository-Editor	Java-Code	C#-Code	Java-Code
	Parameter	✓	✓	✓	✓
	Datenfluss deklariert (Eingabe/Ausgabe)	✓	✓	✓	optional
	Aktivitäten-Dokumentation (AD)	✓	✓	✓	✓
	AD im Modellierungstool verfügbar	✓/–[d]	✓	✓	✓
	Graphisches Icon	✓	–	✓	✓
	Ausgehende Kanten (AK)	–	–	–	✓
	AK in Aktivität definiert	–[e]	–[f]	–	✓
	Benutzungsregeln / Check-Code	–	–	–	LocalChecker
Generierung (von BA)	Stub-Generierung	im Repository-Editor	–	–	SIB-Creator
	vollst. Generierung: ZT[g]	im Repository-Editor	–	–	WSDL2SIB (vollautomatisch)
	vollst. Generierung: ZS[h]	–	–	–	InBuS (halbautomatisch)
Organisation (von BA)	Kategorien	✓	–	–	✓
	hierarchische Kategorien	–	–	–	✓
	lokales Repository	✓	–	–	✓
	remote Repository	✓	–	–	✓/–[i]
Nutzung (von BA)	Gesamtbewertung	+[j]	○[k]	○[l]	+[m]

[a] WSDL2SIB-Generator stattdessen empfohlen
[b] z. B. EXE, GUI, FileIO, StringFormat, XSLT
[c] z. B. IO, GUI, Collections, Office
[d] nur die allgemeine Beschreibung, nicht für Parameter
[e] stattdessen XOR-Activity
[f] stattdessen Aktivität für Exclusive-Choice-Pattern
[g] zielgerichtet auf Technologie
[h] zielgerichtet auf System
[i] einfaches SVN-Repository mit SIBs und Taxonomien
[j] Repository, Dokumentation (nicht im Editor einsehbar), Icons, expliziter Datenfluss
[k] Dokumentation, kein Icon, kein Repository, keine Kategorien
[l] Dokumentation, Icon, kein Repository, keine Kategorien
[m] Taxonomien, Dokumentation, Icons, LocalChecker

KAPITEL 7

Übertragung des InBuS-Konzeptes auf andere BPMS

Durch die Trennung von Backend und Frontend in InBuS ist eine Übertragung auf andere BPMS als das jABC sehr einfach möglich. Dabei kann die Backend-Seite komplett übernommen werden. Nur das Frontend muss an die entsprechenden Prozesskomponenten angepasst werden. Dabei ist es sehr vorteilhaft, dass hier Templates als technologische Grundlage verwendet werden. Existieren schon Templates für z. B. SIBs, so können diese durch eine entsprechende Abwandlung recht schnell zu Templates alternativer Prozesskomponenten werden. In einem ersten Schritt kann dazu einfach eine spezielle Instanz einer Prozesskomponente von Hand implementiert werden. Diese Implementierung dient dann als Vorlage für das neue Template. Man analysiert dabei die Implementierung dahingehend, welche Bestandteile konstant und welche variabel sind. Die variablen Bestandteile werden dann entsprechend durch Konstrukte der Template-Sprache ersetzt. Welche Konstrukte (z. B. konkrete Variablen) zur Verfügung stehen und auch benutzt werden müssen, kann dabei sehr einfach z. B. einem existierenden SIB-Template[1] entnommen werden.

In Listing 7.1 ist eine vereinfachte Version des Templates für Read-Operationen in Microsoft Dynamics NAV dargestellt. Es handelt sich in dem Sinne um eine vereinfachte Version, dass hier zum einen lediglich die SIB-Klasse selbst dargestellt wird, nicht der dazugehörige Service-Adapter, zum anderen, dass hier viele Features ausgelassen wurden, wie zum Beispiel Dokumentation, Referenzierung des Icons, LocalChecker-

[1] oder auch einem Template für beliebige andere Prozesskomponenten

Regeln und die Unterstützung von Code-Generierung durch Genesys. Für die prinzipielle Erklärung des Prinzips und im Sinne der Einfachheit reicht dieses Beispiel jedoch vollkommen aus. Als Template-Sprache kommt hier (wie in Abschnitt 5.3.5 beschrieben) StringTemplate zum Einsatz. In der ersten Zeile ist zu sehen, dass in dieser Datei nicht nur ein Template, sondern eine Template-Gruppe definiert wird. Durch die Verwendung von Template-Gruppen ist es möglich, innerhalb einer Datei verschachtelte Templates einzusetzen, also Templates die in ihrem Innern wieder andere Templates aufrufen (die in der selben Datei definiert sind). Der Einfachheit halber werden bei InBuS grundsätzlich immer Template-Gruppen eingesetzt, auch wenn keine weitere Verschachtelung notwendig ist. So ist es möglich, alle Templates (bzw. hier Template-Gruppen) einheitlich anzusprechen. In Zeile 3 beginnt die Definition des Haupt-Templates, welches in doppelte Spitze Klammern eingefasst wird. Es endet also in Zeile 45. In Zeile 5 wird zum ersten Mal eine Ersetzungsvariable benutzt. Diese sind in Dollarzeichen eingefasst und tauchen im ganzen Template immer wieder auf. Sie sind folgendermaßen zu interpretieren:

- **`$objectLC$`**: object name lower case (der Objektname in Kleinbuchstaben, z. B. „`salesorder`“)
- **`$object1stUC$`**: object name first letter upper case (der Objektname beginnend mit einem Großbuchstaben, z. B. „`SalesOrder`“)
- **`$object1stLC$`**: object name first letter lower case (der Objektname beginnend mit einem Kleinbuchstaben, z. B. „`salesOrder`“)
- **`$date$`**: das aktuelle Datum

Listing 7.1: SIB-Template NAV-Read-Operationen (vereinfacht)

```
1  group PageReadSIBTemplate;
2  
3  PageReadSIBTemplate_Main() ::= <<
4  
5  package de.jabc.dynnav.sibs.$objectLC$;
6  
7  // weitere Import-Statements ausgelassen
8  
9  import de.jabc.dynnav.adapters.$objectLC$.
10        $object1stUC$PageReadServiceAdapter;
11 
12 @SIBClass("InBuS/MSDynNAV/Page/$object1stUC$/Read")
13 @Generated(value="de.jabc.inbus", date="$date$")
14 public class $object1stUC$PageRead
15         implements Executable {
16 
```

```
17      public ContextKey $object1stLC$NumberKey =
18          new ContextKey("$object1stLC$Number",
19                         ContextKey.Scope.DECLARED,
20                         true);
21
22      public ContextKey $object1stLC$Key =
23          new ContextKey("$object1stLC$",
24                         ContextKey.Scope.DECLARED,
25                         true);
26
27      public static String[] BRANCHES =
28          {"default", "error"};
29
30      public String trace(ExecutionEnvironment env) {
31          try {
32              return
33                  $object1stUC$PageReadServiceAdapter.
34                      read(env,
35                           $object1stLC$NumberKey,
36                           $object1stLC$Key);
37              } catch (Throwable e) {
38                  e.printStackTrace();
39              return "error";
40          }
41      }
42  }
43  >>
```

Werden nun die Variablen von der Template-Engine durch ihre Werte ersetzt, sieht das Ergebnis wie in Listing 7.2 aus. Dieses SIB liest einen Verkaufsauftrag „Sales Order“ passend zu einer vorgegebenen Auftragsnummer.

Listing 7.2: Generiertes SIB „Read Sales Order“ (vereinfacht)

```
1   package de.jabc.dynnav.sibs.salesorder;
2
3   // weitere Import-Statements ausgelassen
4
5   import de.jabc.dynnav.adapters.salesorder.
6          SalesOrderPageReadServiceAdapter;
7
8   @SIBClass("InBuS/MSDynNAV/Page/SalesOrder/Read")
9   @Generated(value="de.jabc.inbus", date="06/30/2013")
10  public class SalesOrderPageRead
11          implements Executable {
12
```

```
    public ContextKey salesOrderNumberKey =
        new ContextKey("salesOrderNumber",
                       ContextKey.Scope.DECLARED,
                       true);

    public ContextKey salesOrderKey =
        new ContextKey("salesOrder",
                       ContextKey.Scope.DECLARED,
                       true);

    public static String[] BRANCHES =
        {"default", "error"};

    public String trace(ExecutionEnvironment env) {
        try {
            return
                SalesOrderPageReadServiceAdapter.
                    read(env,
                         salesOrderNumberKey,
                         salesOrderKey);
            } catch (Throwable e) {
                e.printStackTrace();
            return "error";
        }
    }
}
```

Die eigentliche Implementierung des SIBs ist in andere Klassen ausgelagert. Direkt referenziert ist der Service-Adapter[2]. Das entsprechende Template ist relativ einfach, da der Aufruf einfach an den so genannten „LightweightServiceAdapter“ weitergegeben wird[3]. Dieser enthält nun die eigentliche jABC-spezifische Implementierung. Hier wird also die Objektnummer aus dem Kontext gelesen, der Service aufgerufen und anschließend das Ergebnis wieder in den Kontext geschrieben.

Listing 7.3: Template für LightweightServiceAdapter

```
group PageReadLWServiceAdapterTemplate;

PageReadLWServiceAdapterTemplate_Main() ::= <<

package de.jabc.dynnav.adapters.$objectLC$;

import de.jabc.sibs.dynnav.BusinessObject;

```

[2] „SalesOrderPageReadServiceAdapter“
[3] ein Konstrukt, welches für die Genesys-Code-Generierung benötigt wird

```
// weitere Import-Statements ausgelassen

@Generated(value="de.jabc.inbus", date="$date$")
public class $object1stUC$PageReadLWServiceAdapter {

    public static String read(
        LightweightExecutionEnvironment env,
        ContextKeyFoundation $object1stLC$NumberKey,
        ContextKeyFoundation $object1stLC$Key) {

        String $object1stLC$Number =
          (String) env.getLocalContext().
            get($object1stLC$NumberKey);

        BusinessObject $object1stLC$Object =
          $object1stUC$JavaService.
            read$object1stUC$($object1stLC$Number);

        environment.getLocalContext().put(
          $object1stLC$Key, $object1stLC$Object);

            return "default";
    }
}
>>
```

Auch in diesem Template werden wieder nur die gleichen wenigen Variablen benutzt und außer der einfachen Variablenersetzung keine weiteren Features der Template-Engine. Die eigentliche Komplexität steckt im Template des „JavaServices“. Diese Klasse, die vom „LightweightServiceAdapter“ in diesem Beispiel in den Zeilen 23 bis 25 aufgerufen wird, enthält den Code, der den eigentlichen Webservice aufruft, die von JAX-WS generierten Klassen benutzt und aus den Ergebnissen des Service-Aufrufs das „BusinessObject“ zusammenbaut. Hier werden also noch einige weitere Variablen, zum Beispiel für die Bestandteile des BusinessObjects benötigt. Außerdem werden hier komplexere Mechanismen der Template-Engine (bedingte Ausführungen und Schleifen) eingesetzt, um die Menge der Parameter zu behandeln und je nach Komplexitätsgrad der Datenstrukturen und nach den Vorgaben, die bei der Benutzung des Wizards definiert worden sind, die Daten korrekt zu transformieren. Sehr vorteilhaft ist hier, dass die JavaService-Klasse zur Backend-Seite der generierten Klassen gehört, also bei einem Umstieg von jABC auf ein alternatives BPMS wiederverwendet werden kann.

Das Ergebnis der Generierung des LightweightServiceAdapters ist in Listing 7.4 zu sehen.

Listing 7.4: Generierter LightweightServiceAdapter

```
package de.jabc.sibs.dynnav.adapters.salesorder;

import de.jabc.sibs.dynnav.BusinessObject;

// weitere Import-Statements ausgelassen

@Generated(value="de.jabc.inbus", date="$date$")
public class SalesOrderPageReadLWServiceAdapter {

    public static String read(
        LightweightExecutionEnvironment env,
        ContextKeyFoundation salesOrderNumberKey,
        ContextKeyFoundation salesOrderKey) {

        String salesOrderNumber =
          (String) env.getLocalContext().
          get(salesOrderNumberKey);

        BusinessObject salesOrderObject =
          SalesOrderJavaService.
            readSalesOrder(salesOrderNumber);

        environment.getLocalContext().put(
          salesOrderKey, salesOrderObject);

            return "default";
    }
}
>>
```

In Abschnitt 7.1 wird gezeigt, wie entsprechende Templates für jBPM 5 aussehen, in Abschnitt 7.2 wird auf Activiti eingegangen.

7.1 jBPM (ab Version 5.x)

In Listing 7.5 ist das Template für die WID-Datei zu sehen. Man kann erkennen, dass auch hier wieder nur die gleichen Variablen benutzt werden und ausschließlich Variablenersetzung eingesetzt wird.

Listing 7.5: Template jBPM-WID-Definition

```
import org.drools.process.core.
         datatype.impl.type.IntegerDataType;
import org.drools.process.core.
```

```
4           datatype.impl.type.ObjectDataType;
5  [
6    [
7      "name" : "Read$object1stUC$",
8      "parameters" : [
9        "$object1stLC$Number" : new IntegerDataType(),
10       "$object1stLC$" : new ObjectDataType(),
11     ],
12     "displayName" : "Read$object1stUC$",
13     "icon" : "icons/dynnav.png"
14   ]
15 ]
```

In Listing 7.6 ist das Template für die Implementierung eines jBPM-WID dargestellt. Dies ist aus Sicht der Template-Engine ebenso einfach wie alle obigen Templates. Es ähnelt vom Inhalt der ausführenden Methode her auch sehr stark dem Template des LightweightServiceAdapters in Listing 7.3. Auch hier wird wieder die gleichen drei Schritte vollzogen:

1. Nummer aus dem Kontext lesen
2. JavaService aufrufen
3. Ergebnis in den Kontext schreiben

Schritt 2 ist dabei sogar identisch zum LightweightServiceAdapter. Für die Übertragung der Templates auf alternative BPMS sind also lediglich die BPMS-spezifischen Eigenheiten anzupassen.

Listing 7.6: Template jBPM-WID-Implementierung

```
1  package de.example.jbpm.dynnav.$objectLC$;
2
3  import java.util.HashMap;
4
5  import org.drools.runtime.process.WorkItem;
6  import org.drools.runtime.process.WorkItemHandler;
7  import org.drools.runtime.process.WorkItemManager;
8
9  public class Read$object1stUC$NodeImpl
10         implements WorkItemHandler {
11
12    @Override
13    public void abortWorkItem(WorkItem wi,
14                              WorkItemManager wim) {
15    }
16
17    @Override
18    public void executeWorkItem(WorkItem wi,
```

```
19                                          WorkItemManager wim) {
20
21          Integer $object1stLC$Number = (Integer)
22            wi.getParameter("$object1stLC$Number");
23
24          BusinessObject $object1stLC$Object =
25            $object1stLC$JavaService.
26              read$object1stUC$($object1stLC$Number);
27
28          HashMap<String, Object> results =
29            new HashMap<String, Object>();
30
31          results.put("$object1stLC$", "bla");
32            wim.completeWorkItem(wi.getId(), results);
33      }
34  }
```

7.2 Activiti

Die JavaDelegates von Activiti sind ebenso einfach mit entsprechenden Templates zu generieren. Ein Beispiel-Template ist in Listing 7.7 dargestellt, das generierte Ergebnis in Listing 7.8.

Listing 7.7: Template JavaDelegate

```
1   package de.activiti.test;
2
3   import org.activiti.engine.delegate.DelegateExecution;
4   import org.activiti.engine.delegate.Expression;
5   import org.activiti.engine.delegate.JavaDelegate;
6
7   public class $object1stUC$CustomService implements
8       JavaDelegate {
9
10      private Expression $object1stLC$NumberKey;
11      private Expression $object1stLC$Key;
12
13      @Override
14      public void execute(DelegateExecution exec)
15              throws Exception {
16
17          Integer $object1stLC$Number = (Integer)
18            exec.getVariable((String)
19              $object1stLC$NumberKey.getValue(exec));
20
```

```
21          BusinessObject $object1stLC$Object =
22            $object1stUC$JavaService.
23              read$object1stUC$($object1stLC$Number);
24
25          execution.setVariable((String)
26            $object1stLC$Key.getValue(execution),
27            $object1stLC$);
28      }
29
30  }
```

Listing 7.8: Template JavaDelegate

```
1   package de.activiti.test;
2
3   import org.activiti.engine.delegate.DelegateExecution;
4   import org.activiti.engine.delegate.Expression;
5   import org.activiti.engine.delegate.JavaDelegate;
6
7   public class SalesOrderCustomService implements
8       JavaDelegate {
9
10      private Expression salesOrderNumberKey;
11      private Expression salesOrderKey;
12
13      @Override
14      public void execute(DelegateExecution exec)
15              throws Exception {
16
17          Integer salesOrderNumber = (Integer)
18            exec.getVariable((String)
19              salesOrderNumberKey.getValue(exec));
20
21          BusinessObject salesOrderObject =
22            SalesOrderJavaService.
23              read$object1stUC$(salesOrderNumber);
24
25          execution.setVariable((String)
26            salesOrderKey.getValue(execution),
27            salesOrder);
28      }
29
30  }
```

Hier gilt das gleiche wie für die Übertragung auf jBPM5. Jedoch ist hier zusätzlich zu beachten, dass es sich bei Activiti nicht um einfach wiederverwendbare Business-Aktivitäten handelt. Wenn dies erreicht werden

sollte, so müsste der Prozess-Editor von Activiti noch mindestens um folgende Features erweitert werden:

- Auflistung aller JavaDelegates in Kategorien.
- Editor für die Kategorisierung der JavaDelegates.
- Automatische Erkennung der Attribute der JavaDelegates[4].
- Anzeige von Dokumentation zu JavaDelegates im Prozess-Editor[5]

[4]so dass der Prozess-Modellierer nicht in die Implementierung des JavaDelegates hinein sehen muss

[5]Dazu müsste auch noch eine Stelle definiert werden, an der die Dokumentation hinterlegt werden kann.

KAPITEL 8

Einfachheit als Erfolgsfaktor im BPM

Wie schon in Abschnitt 2.7 beschrieben, spielt Einfachheit im Bereich des Geschäftsprozessmanagements eine wichtige Rolle. Die Forderung nach Einfachheit betrifft hier verschiedenste Bereiche. Im Folgenden wird auf drei verschiedene Bereiche eingegangen:

- Einfachheit bei der Service-Integration
- Einfachheit der Prozesssprachen
- Einfachheit durch Abstraktion: „BPM in der Cloud“

Der erste Bereich (Abschnitt 8.1) betrifft das zentrale Thema dieser Arbeit, die Service-Integration. Dabei wird noch einmal kurz zusammengefasst, wie hier Einfachheit erreicht wird. In Abschnitt 8.2 wird aufgezeigt, dass auch die Prozessbeschreibungssprachen einfach sein sollten und Beispiele aufgezeigt, in denen dies nicht der Fall ist. Abschnitt 8.3 befasst sich schließlich damit, wie die Prozessmanagementsoftware möglichst ohne Administrationsaufwand und eigenes Rechenzentrum betrieben wird - es geht um „Cloud-Computing“. BPM in der Cloud ist ein aktueller Trend, der sehr vielversprechend ist. Gerade hier entstehen im Moment Produkte, bei denen ebenfalls Einfachheit im Mittelpunkt steht.

8.1 Einfachheit bei der Service-Integration

Wie in Kapitel 4 beschrieben, kann die Einbindung von Services in ausführbare Geschäftsprozesse auf unterschiedliche Arten erfolgen. Entsprechend unterschiedlich einfach ist dabei das entsprechende Vorgehen. Aus

Sicht des Prozessmodellierers ist sicherlich die Benutzung der domänenspezifischen Business-Aktivitäten am einfachsten. Hier muss er sich keine Gedanken um die zu Grunde liegende Technologie machen und sich entsprechend mit diesen Details nicht auskennen. Er kann einfach aus einer Palette von konkreten fachlichen Services den richtigen auswählen und in seinem Prozessmodell platzieren. Technische Service-Aktivitäten oder Skript-Aktivitäten sind dagegen sehr technisch und kaum vom Anwendungsexperten zu benutzen.

Damit Business-Aktivitäten wirklich einfach zu benutzen sind, sollten Sie folgende Eigenschaften besitzen:

- Sie sollten einen aussagekräftigen Namen besitzen.
- Sie sollten in eine passende Kategorie eingeordnet sein, damit man sie auch in einer großen Sammlung gut finden kann.
- Sie sollten gut und ausführlich dokumentiert sein.
- Sie sollten die richtige Granularität besitzen: Eine längere Folge von Aktivitäten für einen Service-Aufruf ist keine gute Granularität. Ebenso möchte man aber auch nicht eine Aktivität für einen ganzen lang dauernden Prozess, der eigentlich noch viele entscheidende Zwischenzustände besitzt.
- Ein gutes, illustrierendes Icon ist von Vorteil.
- Sinnvolle Benutzungsregeln, die während der Modellierung automatisch ausgewertet werden können, helfen bei der korrekten Verwendung.

Da Business-Aktivitäten immer für konkrete Services erstellt werden, ist hier besonders zu beachten, dass es ratsam ist, diese erst bei Bedarf zu erstellen. Dabei ist es egal, ob sie durch einen Generator oder von Hand erstellt werden. In jedem Fall führt eine Produktion „auf Halde“ zu viel überflüssiger Arbeit. Bei der Erstellung der Business-Aktivitäten ist eine intensive Kommunikation zwischen Anwendungsexperten und IT-Experten unerlässlich.

8.2 Einfachheit der Prozesssprachen

Bei der Auswahl einer Prozessbeschreibungssprache ist von enormer Wichtigkeit, dass diese möglichst einfach zu verstehen ist. Nur so kann der Modellierer gute Ergebnisse erzielen. Die Prozessmodelle sind immer auch Diskussionsgrundlage für die verschiedenen involvierten Parteien (wie Management, IT-Abteilung, Anwender, Kunden), wobei wichtig ist, dass das Modell alle gleich verstehen. Oft ist es jedoch aufgrund der Komplexität der Sprachen der Fall, dass entweder kaum jemand das Modell

versteht oder zwar alle der Meinung sind, es zu verstehen, es jedoch erhebliche Abweichungen in der Interpretation gibt. Besonders, wenn das Prozessmodell durch eine Engine ausgeführt werden soll, ist ein korrektes Verständnis der Semantik unerlässlich. Aktuelle Prozessbeschreibungssprachen sind hier bezüglich der Einfachheit nicht immer vorbildlich. Im Folgenden wird betrachtet, wo konkrete Probleme bei BPMN, AristaFlow, YAWL und BPEL liegen.

BPMN: BPMN ist eine sehr komplexe Sprache mit einer großen Anzahl unterschiedlicher Elemente. Ein gutes Beispiel ist der Vorrat an unterschiedlichen Ereignistypen. Diese sind in einer Matrix in Abbildung 8.1 dargestellt. Viele dieser Ereignistypen werden kaum oder gar nicht in der Modellierung verwendet. Bei den Engines ist die Situation sogar so, dass aktuell meist nur ein kleiner Bruchteil der Events unterstützt wird. Wird die Sprache in ihrer gesamten Komplexität genutzt, ist der Prozess kaum noch zu verstehen. In Abbildung 8.2 ist ein BPMN-Kollaborationsdiagramm[1] zu sehen, welches die Verwendung unterschiedlicher Elemente demonstrieren soll. Diese Abbildung stammt wie Abbildung 8.1 aus einem Poster von mehreren Herstellern von BPMN-2.0-Tools. In beiden Abbildungen wird sehr gut deutlich, wie komplex die Aufgabe für den Modellierer[2] ist, zu jedem Element die richtige Semantik zu kennen.

Ein weiteres Problem bei BPMN ist, dass der gleiche Sachverhalt oft auf sehr unterschiedliche Arten dargestellt werden kann. Wenn man BPMN verwendet, ist es daher oft üblich, sich zunächst innerhalb der Sprache auf Konventionen zu einigen, auf welche Art bestimmte Muster abgebildet werden. Es wird also ein „Methoden- und Konventionenhandbuch“ erstellt. Folgendes sind Beispiele für unterschiedliche Modellierungen gleicher Sachverhalte:

- Exklusive Entscheidungen können durch ein entsprechendes XOR-Gateway (ein rautenförmiger Knoten mit einem X) oder durch mehrere bedingte, ausgehende Kanten (mit kleinen Rauten an der Quellseite) modelliert werden.
- Parallelität kann einfach durch mehrere ausgehende Kanten oder explizit durch ein AND-Gateway dargestellt werden. Eine dritte mögliche Variante ist hier die Platzierung mehrerer, nicht verbundener Tasks innerhalb eines Teilprozesses.

[1] Ein Kollaborationsdiagramm ist ein BPMN-Diagramm, welches potentiell mehrere miteinander kommunizierende Prozesse (jeweils in eigenen Pools) darstellt. Die kommunizierenden Prozesse können dabei auch abstrakt als leerer Pool („Black Box“) dargestellt werden.
[2] und für alle, die den Prozess lesen und verstehen sollen

	Start			Zwischen				Ende
	Standard	Ereignis-Teilprozess Unterbrechend	Ereignis-Teilprozess Nicht-unterbrechend	Eingetreten	Angeheftet unterbrechend	Angeheftet Nicht-unterbrechend	Ausgelöst	Standard
Blanko: Untypisierte Ereignisse, i. d. R. am Start oder Ende eines Prozesses.								
Nachricht: Empfang und Versand von Nachrichten.								
Timer: Periodische zeitliche Ereignisse, Zeitpunkte oder Zeitspannen.								
Eskalation: Meldung an den nächsthöheren Verantwortlichen.								
Bedingung: Reaktion auf veränderte Bedingungen und Bezug auf Geschäftsregeln.								
Link: Zwei zusammengehörige Link-Ereignisse repräsentieren einen Sequenzfluss.								
Fehler: Auslösen und behandeln von definierten Fehlern.								
Abbruch: Reaktion auf abgebrochene Transaktionen oder Auslösen von Abbrüchen.								
Kompensation: Behandeln oder Auslösen einer Kompensation								
Signal: Signal über mehrere Prozesse. Auf ein Signal kann mehrfach reagiert werden.								
Mehrfach: Eintreten eines von mehreren Ereignissen. Auslösen aller Ereignisse.								
Mehrfach/Parallel: Eintreten aller Ereignisse.								
Terminierung: Löst die sofortige Beendigung des Prozesses aus.								

Abbildung 8.1: BPMN 2.0 Ereignisse (Quelle: [BPM11])

- Der Empfang bzw. der Versand von Nachrichten kann durch entsprechende Events oder durch entsprechend markierte Tasks modelliert werden.

AristaFlow: Das Prozessmodell von AristaFlow kann schon bei relativ einfachen Prozessen zu komplexen Darstellungen führen. Abbildung 8.3 stammt aus einem AristaFlow-Workshop, der von den Herstellern selbst angeboten wurde. Dargestellt ist ein Prozess, bei dem eingangs nach einer Zahl gefragt wird und anschließend in einer einfachen Schleife diese Zahl abwechselnd immer wieder ausgegeben wird und anschließend um 1 verringert wird, bis die Zahl 0 erreicht ist. Die hier zu sehende Darstellung ist leicht gegenüber dem Original abgewandelt. Die erste Abweichung besteht darin, dass hier das Datenobjekt, also die Integer-Variable, nicht

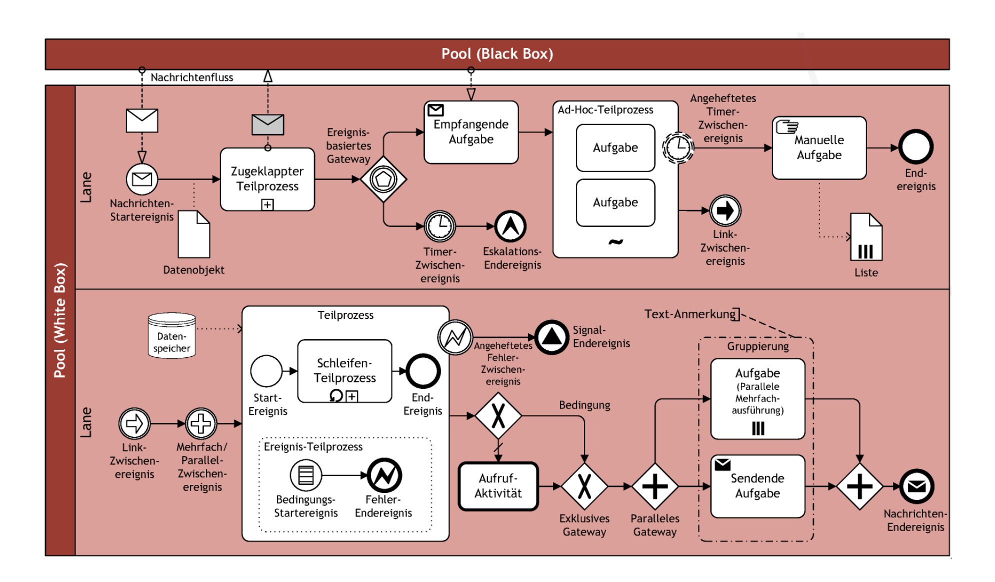

Abbildung 8.2: BPMN 2.0 Kollaborationsdiagramm (Quelle: [BPM11])

zu sehen ist. Bei AristaFlow ist es üblich, auch die Datenobjekte explizit als Knoten darzustellen. Wird dieser Variable ein Wert zugewiesen, so wird dies durch eine gerichtete Kante von der entsprechenden Aktivität zum Datenobjekt hin dargestellt, wird der Wert gelesen, so existiert eine gerichtete Kante vom Datenobjekt zur lesenden Aktivität. Diese Datenobjekte und die damit verbundenen Kanten lassen sich im Tool optional auch ausblenden. Hier wurde auf die Darstellung verzichtet, da nur gezeigt werden soll, dass auch die Darstellung des Kontrollflusses allein schon zu komplex ist. Die zweite Abweichung ist das Layout. AristaFlow besitzt einen permanenten und automatischen Layouter, der dafür sorgt, dass bei einem Einfügen eines neuen Knotens dieser immer an der gleichen, dafür vorgesehenen Stelle eingefügt wird. Somit hat der Modellierer diesbezüglich keine Freiheiten. Sequenzen werden hier immer in einer Reihe von links nach rechts dargestellt. In dieser Abbildung wurde hier lediglich am Anfang und am Ende des Prozesses davon abgewichen, um eine kompaktere Darstellung zu gewinnen[3].

Es gibt mehrere Aspekte, welche die Darstellung der Schleife komplex machen:

- Schleifen sind nur durch die Benutzung einer entsprechenden Aktivität erlaubt. Dabei gibt nur Repeat-Until-Schleifen[4]. In Abbildung 8.3 ist die Schleifen-Aktivität mit „fertig ?“ beschriftet[5]. Der Rück-

[3]Der Prozess im Original-Layout wäre im Ganzen so breit gewesen, dass die Beschriftungen nicht zu lesen wären.

[4]Eine Schleife wird also immer mindestens einmal durchlaufen und die Überprüfung der Schleifenbedingung erfolgt immer am Ende einer Iteration

[5]Die Schleifen-Aktivität ist an dem Kreis an der ausgehenden Kante zu erkennen

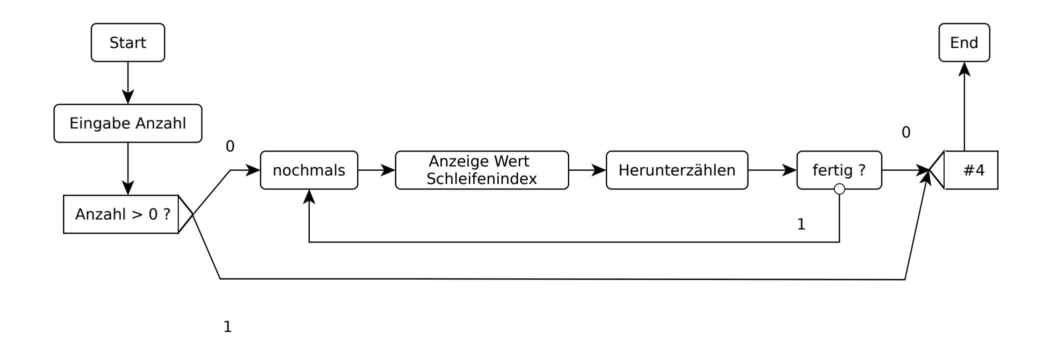

Abbildung 8.3: Eine Schleife in AristaFlow

sprung nach der Überprüfung der Schleifenvariable erfolgt hier zu einer Aktivität, die selbst nichts ausführt (hier mit „nochmals“) beschriftet. Diese wäre in einem SLG nicht nötig. Durch die Beschränkung auf Repeat-Until-Schleifen ist es in diesem Fall nötig, die Bedingung vor der Schleife noch einmal abzufragen. Es wurde also eine XOR-Aktivität (mit der Beschriftung „Anzahl > 0 ?“) eingefügt, welche die gleiche Bedingung enthält wie die Schleifen-Aktivität. Hier erfolgt also eine Dopplung des Arbeits- und Wartungsaufwandes. Wenn die Bedingung geändert werden soll, so muss immer daran gedacht werden, dies an beiden Stellen zu tun. Außerdem ist diese Dopplung für das Verständnis dieses Prozesses nicht förderlich.

- Die verwendete XOR-Aktivität benötigt zur späteren Zusammenführung der Kontrollflüsse ein entsprechendes Pendant (ein XOR-Join mit der Beschriftung „#4“. Die Beschriftung hat hier keine Bedeutung, sondern entspricht lediglich dem Default-Wert des vierten eingefügten Knotens. Zum einen wäre es schöner, wenn dieser Knoten gar nicht nötig wäre (wie in einem SLG), zum anderen wäre zu wünschen, dass nichtssagende Beschriftungen auch weggelassen werden dürfen.
- Die ausgehenden Kanten der XOR-Aktivität besitzen hier die Beschriftungen 0 und 1, was den Wahrheitswerten „falsch“ und „wahr“ entspricht. Diese Beschriftungen sind fest vorgegeben. Sie sind leider wenig intuitiv für Fachanwender. Sprechende Bezeichner wie „ja“ und „nein“ währen hier angebrachter.
- Es existieren explizite Startknoten und Endknoten, die sonst nichts machen und auch immer die jeweils gleiche feste Bezeichnung tragen. In BPMN können diese noch weitere Informationen tragen und mit individuellen Bezeichnungen versehen werden, so dass diese Elemente sinnvoll zum Verständnis beitragen können. Im jABC sind

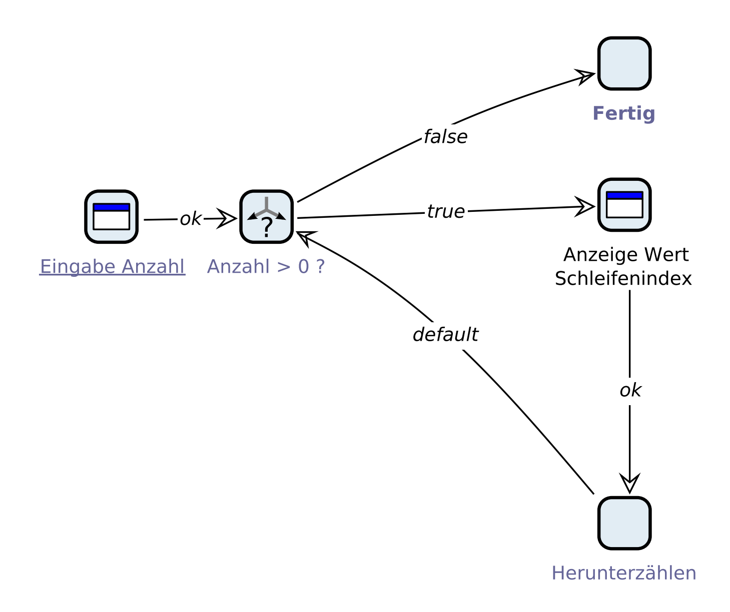

Abbildung 8.4: Die gleiche Schleife im jABC

> eigene Start- und Endereignis optional. Tragen sie nicht zu einem besseren Verständnis bei, so können sie einfach weggelassen werden. Als Startereignis kann jedes beliebige SIB gekennzeichnet werden[6], ein Endereignis besitzt im Allgemeinen einen ausgehenden Modelbranch[7].

In Abbildung 8.4 ist dargestellt, wie die Schleife als SLG im jABC modelliert wird. Die Anzahl der Knoten verringert sich dabei von neun auf fünf. Eigentlich würden hier auch vier Knoten reichen, da der Endknoten („Fertig“) optional ist. Dieser wurde lediglich eingefügt, um noch deutlicher anzuzeigen, dass von dem Verzweigungsknoten aus der Prozess auch verlassen werden kann.

YAWL: YAWL ist eine Sprache, bei der das Ziel ist, alle existierenden Workflow-Patterns direkt zu unterstützen. Noch weitaus extremer als bei BPMN führt dies dazu, dass die Anzahl verschiedener Elemente unüberschaubar groß ist und es somit sehr aufwändig ist, die Sprache zu lernen und komplett zu beherrschen.

BPEL: BPEL hat eine sehr technische Natur und orientiert sich eng an den Web-Service- und XML-Technologien. Die schon in Abschnitt

[6]zu erkennen am unterstrichenen Label
[7]zu erkennen an einem fett dargestellten Label

6.7 erwähnten Erkenntnisse aus eigenen Arbeiten sowie verschiedener Diplomarbeiten haben gezeigt, dass BPEL weit davon entfernt ist, das ITSy-Prinzip zu unterstützen.

8.3 Einfachheit durch Abstraktion: „BPM in der Cloud“

Soll in einem Unternehmen ein durchgängiges Geschäftsprozessmanagement eingeführt werden, so ist es grundsätzlich schon mit einem erheblichen Aufwand verbunden, dafür nur die technische Grundlage zu schaffen. An den Arbeitsplätzen müssen entsprechende Modellierungstools installiert werden, für die Ausführung muss eine Prozess-Engine auf einem Server laufen und diese Engine benötigt meist noch einen Applikationsserver[8] als Basis. Dazu können noch weitere Applikationen zur Kommunikation und Koordination der beteiligten Abteilungen und Personen kommen, wie z. B. Aufgabenlisten-Verwaltungen oder Kalender. All diese Applikationen müssen dabei initial installiert und konfiguriert werden und anschließend auch kontinuierlich aktualisiert und bei auftretenden Fehlern oder Sicherheitslücken entsprechend gepatcht werden. Für den Betrieb (besonders der Server-Software) dabei meist ein Rechenzentrum betrieben mit einer Reihe von Servern, die sicher verschlossen, bedacht und klimatisiert werden müssen. Weiterhin muss durch entsprechende Redundanzen und Notstromversorgungen für die notwendige Sicherheit und Verfügbarkeit gesorgt werden. All dies zusammen bedeutet einen erheblichen Aufwand für eine IT-Abteilung und entsprechend hohe Kosten für das Unternehmen. Oft ist es daher ratsam, all dies auszulagern und auch hier auf Cloud-Computing zu setzen. So muss sich die eigene IT-Abteilung um die erwähnten Aufgaben nicht kümmern und die Ressourcen werden im Allgemeinen deutlich besser ausgelastet. Das Unternehmen bezahlt in dem Fall nur das, was wirklich genutzt wird.

„BPM in der Cloud“ ist folglich ein Trend, der aktuell immer mehr zu einem Hype wird. In einer entsprechenden Projektgruppe namens „PG PCB: Process Cloud for Business“ wurde evaluiert, wie gut es heute möglich ist, existierende Prozess-Engines auf verschiedenen IaaS- oder PaaS-Angeboten auszuführen. Zum einen wurde die „Google App Engine“[9] betrachtet, ein PaaS-Angebot von Google, welches unter anderem erlaubt Java-basierte Web-Anwendungen zu deployen. Dabei existieren hier eine ganze Reihe von Einschränkungen für die Applikationen. Als Datenbank für die NoSQL-Datenbank BigTable von Google selbst genutzt, ande-

[8] z. B. einen JavaEE-Applikationsserver
[9] https://developers.google.com/appengine/

re Datenbanken (z. B. relationale über JDBC) sind nicht erreichbar. Es ist weder erlaubt, eigene Threads zu starten noch auf das Dateisystem zuzugreifen. Eine weitere Einschränkung besteht darin, dass nicht alle Java-Klassen des JRE genutzt werden dürfen, sondern nur ca. 40% aller Klassen, die in einer entsprechenden Whitelist aufgezählt werden. All diese Einschränkungen machen es schwierig bis unmöglich, existierende Engines wie jABC oder Activiti zu deployen. Diese Erfahrung machte auch die Projektgruppe. Hier wurde intensiv daran gearbeitet, verschiedene Versionen der beiden Engines auf die Google-Plattform zu bringen. Letztlich erfolgreich konnte Activiti auf der PaaS „Heroku“[10] von Salesforce deployt werden, welche deutlich weniger Einschränkungen definiert. Da während der Laufzeit der Projektgruppe klar wurde, dass ein großes Problem im Cloud-Computing in den fehlenden Standards und somit geringer Portabilität und Interoperabilität zwischen den Clouds liegt, wird in der Projektgruppe „Intercloud“ die Arbeit fortgesetzt. Hier soll nun einerseits untersucht werden, wie verschiedene Clouds zusammen benutzt werden können und andererseits wie gemeinsame Schnittstellen definiert werden können, so dass es einfach möglich ist, Applikationen von einer Cloud in die andere zu portieren.

Bei der Verwendung von IaaS wie z. B. Amazon EC2 hat man letztlich keine Einschränkungen, aber auf der anderen Seite auch mehr Aufwand, da hier das Laufzeitsystem selbst installiert und gewartet werden muss. Die Angebote von Amazon wurden in der Bachelorarbeit von Moussa Oumarou [Oum12] untersucht. Hier wurde jBPM sowohl direkt auf EC2 als auch auf das darauf basierende Java-PaaS „Elastic Beanstalk“[11] deployt. Beides funktionierte ohne Probleme. In umfangreichen Tests wurde gezeigt, dass die Nutzung von Cloud-Computing-Technologien (hier insbesondere von Amazon) eine gute technische Basis für die Ausführung von Geschäftsprozessen sein kann.

Das Deployment existierender Prozess-Engines auf IaaS oder PaaS ist zwar schon ein guter Schritt, wenn von der darunter liegenden Technologie abstrahiert werden soll, noch besser ist es jedoch, eine fertige SaaS-Prozess-Engine einzusetzen. Prozess-Modellierungstools existieren schon länger als SaaS, das bekannteste Beispiel hierfür ist wohl das auch von Activiti eingesetzte Signavio[12]. Doch in letzter Zeit beginnt auch die Zeit der SaaS-Engines. Amazon bietet hier zum Beispiel mit seinem Dienst SWF[13] eine vielversprechende Lösung an[14]. Ein weiteres interessantes

[10] https://www.heroku.com/
[11] http://aws.amazon.com/de/elasticbeanstalk/
[12] http://www.signavio.com/de/
[13] SWF: Simple Workflow Service
[14] http://www.allthingsdistributed.com/2012/02/Amazon-Simple-Workflow-Service.html und http://aws.amazon.com/de/swf/

Projekt stammt von Tom Baeyens, der schon Gründer und Hauptentwickler der Projekte jBPM und Activiti war. Dieser verließ Activiti um mit „Effektif“[15] ein neues Projekt zu starten. Dabei handelt es sich ebenfalls um ein komplett Cloud-basiertes BPM-System, also einer fertigen SaaS-Lösung. Enthalten sind sowohl ein Modellierungstool[16] als auch eine Prozessengine und eine Aufgabenverwaltung. Zum Zeitpunkt der Veröffentlichung dieser Arbeit war die Software selbst noch nicht veröffentlicht. Es existieren lediglich Vorträge und Whitepapers, die beschreiben, wie das fertige Projekt aussehen soll. Bei Effektiv wird besonderer Wert auf Einfachheit gelegt. Einfache und häufig vorkommende Dinge sollen mit wenigen Klicks zu erledigen sein.

[15] http://www.effektif.com/

[16] Hier wird Signavio eingesetzt. Die Signavio GmbH ist auch Hauptunterstützer und Finanzgeber des Projektes.

KAPITEL 9

Fazit und Ausblick

Im Folgenden wird zunächst in Abschnitt 9.1 ein Fazit dieser Arbeit gezogen bevor dann schließlich in Abschnitt 9.2 ein Ausblick auf mögliche Erweiterungen und Fortsetzungen der in dieser Arbeit behandelten Themen gewagt wird.

9.1 Fazit

In dieser Arbeit wurde gezeigt, dass heutige Servicesammlungen oft leider eine enttäuschende Qualität bieten. Dies gilt auch für Produkte namhafter Hersteller. Positiv ist der Trend zu bewerten, den das Cloud-Computing in diesem Bereich angestoßen hat. Hier ist offensichtlich ein echter Nährboden für gute Konzepte zur Realisierung von APIs entstanden. Sollen diese Services nun in den Geschäftsprozessen eingebunden werden, so ist zu beobachten, dass viele BPMS gerade in diesem Gebiet noch sehr wenig Unterstützung bieten. Auch dies wurde ausführlich in dieser Arbeit belegt. Weiterhin wurde gezeigt, wie die Ideen und Konzepte des „Extreme Model Driven Design“ und eine Wizard-gesteuerte Generierung von Prozessaktivitäten besonders bei großen APIs, wie sie für ERP-Systeme typisch sind, die Service-Integration stark vereinfachen können.

Die genannten Punkte wurden durch eine Vielzahl flankierender Tätigkeiten ergänzt, die jeweils ebenfalls mit in diese Arbeit eingeflossen sind:

- Es wurde sich kritisch mit dem Einsatz von Tabellenkalkulationssoftware in Unternehmen befasst. Neben eigenen Arbeiten wurde zu dem Thema auch die Projektgruppe „Excelerate“ durchgeführt. Dies wird in Abschnitt 5.2.1 beschrieben.
- Es wurden zahlreiche manuelle SIB-Implementierungen durchgeführt, zum Beispiel für Office-Produkte (siehe Abschnitt 5.2.1 oder die SAP-BAPI (siehe Abschnitt 5.2.2).
- Sowohl die eigene Software zur Generierung von SIBs als auch die APIs der untersuchten Systeme wurden auch durch Studenten der Vorlesung „Aktuelle Themen der Dienstleistungsinformatik“ evaluiert. In praktischen Projekten wurden dabei auch Implementierungen von SIBs für SAP eSOA und Google Apps durchgeführt. Dies wird in Abschnitt 5.2.3 genauer beschrieben.
- Es wurde sich intensiv mit der Prozessbeschreibungssprache BPEL auseinandergesetzt. Unter anderem wurde die Sprache in mehreren Diplomarbeiten und Projektgruppen eingesetzt (siehe dazu die Abschnitte 6.7 und 8.2).
- Ebenso intensiv wurde die Sprache BPMN evaluiert. So setzen zum Beispiel mehrere untersuchte BPMS BPMN ein. BPMN wird unter anderem in den Abschnitten 4.1, 6.1, 6.2, 6.3 und 8.2 thematisiert.
- Es wurden die Möglichkeiten des Cloud-Computing für BPM erforscht, zum Beispiel in der Projektgruppe „PCB“, in der Projektgruppe „Intercloud“ oder der Bachelorarbeit von Moussa Oumarou [Oum12]. Dies wird in Abschnitt 8.3 behandelt.
- Durch alle Arbeiten zieht sich das Thema „Einfachheit“ als Leitmotiv. Dazu wurde auch von Jan Pardo eine entsprechende Diplomarbeit angefertigt. Auf dieses Thema wird besonders in Abschnitt 2.7 und in Kapitel 8 eingegangen.

9.2 Ausblick

Diese Arbeit hat zum Ziel, ein Leitfaden im Bereich der Service-Integration zu sein, und dies sowohl für die Seite der Service-Anbieter als auch für die Seite der Service-Nutzer. Um noch mehr praktischen Nutzen aus diesem Leitfaden zu ziehen, bieten sich noch sehr viele Möglichkeiten an, diese Arbeit fortzusetzen.

Im Bereich der APIs existieren noch viele weitere Produkte, die es wert wären, untersucht zu werden. Dazu könnten zu Beispiel die folgenden gehören:

- Zahlreiche Produkte der Firma Oracle, unter anderem auch „JD Edwards“
- Das ERP-System „Infor ERP LN“ (ehemals „Baan“)
- Die neben Dynamics NAV weiteren Produkte aus Microsofts Dynamics-Produktreihe, wie z. B. Dynamics AX oder Dynamics GP.

Die Menge der auf dem Markt verfügbaren BPMS ist mittlerweile fast unüberschaubar geworden. Potenziell zu untersuchende Systeme wären hier noch die folgenden:

- Oracle BPM
- IBM Business Process Manager
- SAP Netweaver Business Process Management
- Software AG webMethods
- inubit BPM Suite (was mittlerweile zur Firma Bosch gehört)
- Soreco Xpert.ivy
- Cordys BOP
- AgilePoint

Die Software zur SIB-Generierung kann immer noch weiter verbessert und ausgebaut werden:

- Es sind fortgeschrittenere Funktionen zur Datentransformation denkbar.
- Die Integration bestehender Frameworks zur GUI-Generierung für GUI-SIBs würde die Software weiter komplettieren.
- Die möglichst einfache Definition komplexer Abfragen (besonders an Business-Objekte) wäre ein eigenes Forschungsthema, welches sich an diese Arbeit anschließen könnte.
- Man könnte Generatoren von Prozess-Aktivitäten für alle untersuchten APIs sowie alle untersuchten BPMS erstellen.

Literaturverzeichnis

[Abe99] ABELSON, H.: *Architects of the information society: Thirty-five years of the laboratory for computer science at MIT*. MIT Press, 1999

[ABP08] ANDRIKOPOULOS, Vasilios ; BENBERNOU, Salima ; PAPAZOGLOU, Mike P.: Managing the evolution of service specifications. In: *Advanced Information Systems Engineering* Springer, 2008, S. 359–374

[ACD+12] ADOUAKOU, Ettiboa ; CHRISTIDIS, Georgios ; DREES, Alexander ; GABRIEL, Felix ; HAMMERL, Christian ; LORENZ, Kersten ; MARRO, Donato ; PARDO, Jan ; THALMANN, Carola ; WANG, Hongzhi: PG 551 - Excelerate - Endbericht. (2012)

[Ack10] ACKERMANN, Andre: *Automatische Generierung von Softwarebausteinen zur Modellierung ERP-System übergreifender Geschäftsprozesse*, Universität Potsdam, Diplomarbeit, 2010

[act13] *Activiti Website*. `http://www.activiti.org/`. Version: 2013

[Ada07] ADAMS, Michael J.: *Facilitating dynamic flexibility and exception handling for workflows*, Queensland University of Technology Brisbane, Australia, Diss., 2007

[ADe13] ADEMPIERE E.V.: *ADempiere Webseite*. `http://www.adempiere.com/`. Version: 2013

[Aga04] AGARWAL, Anamika: *Versioning of Web service interfaces*, Massachusetts Institute of Technology, Masterarbeit, 2004

[Ama13] AMAZON: *Amazon Web Services - Website*. http://aws.amazon.com/de/. Version: 2013

[AMM08] AL-MASRI, E. ; MAHMOUD, Q.H.: Investigating web services on the world wide web. In: *Proceedings of the 17th international conference on World Wide Web* ACM, 2008, S. 795–804

[Apa13] APACHE SOFTWARE FOUNDATION: *Apache Ofbiz Webseite*. http://ofbiz.apache.org/. Version: 2013

[Ari13] ARISTAFLOW GMBH: *AristaFlow Website*. http://www.aristaflow.com/. Version: 2013

[BB05] BORE, C. ; BORE, S.: Profiling software API usability for consumer electronics. In: *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on* IEEE, 2005, S. 155–156

[BBCT04] BAINA, Karim ; BENATALLAH, Boualem ; CASATI, Fabio ; TOUMANI, Farouk: Model-driven web service development. In: *Advanced Information Systems Engineering* Springer, 2004, S. 527–543

[Beh06] BEHERA, Gopala K.: *BPM and SOA: A Strategic Alliance*. http://www.bptrends.com/publicationfiles/05-06-WP-BPM-SOA-Behara.pdf. Version: 2006

[BFHM12] BURNS, C. ; FERREIRA, J. ; HELLMANN, T.D. ; MAURER, F.: Usable results from the field of API usability: A systematic mapping and further analysis. In: *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on* IEEE, 2012, S. 179–182

[BGK+04] BLOW, Michael ; GOLAND, Yaron ; KLOPPMANN, Matthias ; LEYMANN, Frank ; PFAU, Gerhard ; ROLLER, Dieter ; ROWLEY, Michael: BPELJ: BPEL for Java., BEA and IBM, March 2004

[Bid00] BIDER: Business Process Modelling - Concepts. In: *PBPM'00*, 2000

[BJM09] BAKERA, Marco ; JÖRGES, Sven ; MARGARIA, Tiziana: Test your strategy: graphical construction of strategies for connect-four. In: *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on* IEEE, 2009, S. 172–181

[BJX+08] BEATON, Jack ; JEONG, Sae Y. ; XIE, Yingyu ; STYLOS, Jeffrey ; MYERS, Brad A.: Usability challenges for en-

terprise service-oriented architecture APIs. In: *VL/HCC*, IEEE, 2008, S. 193–196

[BLHL+01] Berners-Lee, T. ; Hendler, J. ; Lassila, O. u. a.: The semantic web. In: *Scientific american* 284 (2001), Nr. 5, S. 28–37

[Blo06] Bloch, J.: How to design a good API and why it matters. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* ACM, 2006, S. 506–507

[BLWG99] Bisbal, Jesús ; Lawless, Deirdre ; Wu, Bing ; Grimson, Jane: Legacy information systems: Issues and directions. In: *Software, IEEE* 16 (1999), Nr. 5, S. 103–111

[BMRS07] Bakera, Marco ; Margaria, Tiziana ; Renner, Clemens D. ; Steffen, Bernhard: Verification, diagnosis and adaptation: Tool supported enhancement of the model-driven verification process. In: *Revue des Nouvelles Technologies de Information (RNTI-SM-1)* (2007), S. 85–98

[BMRS09] Bakera, M. ; Margaria, T. ; Renner, C.D. ; Steffen, B.: Tool-supported enhancement of diagnosis in model-driven verification. In: *Innovations in Systems and Software Engineering* 5 (2009), Nr. 3, S. 211–228

[BMS+08] Beaton, Jack ; Myers, Brad A. ; Stylos, Jeffrey ; Jeong, Sae Y. ; Xie, Yingyu: Usability Evaluation for Enterprise SOA APIs. In: *in 2nd International Workshop on Systems Development in SOA Environments, SDSOA 2008 (Co-located with ICSE*, 2008, S. 29–34

[Bon13] BonitaSoft: *Bonita Open Solution Website*. http://www.bonitasoft.com/products/bonita-open-solution-open-source-bpm. Version: 2013

[BPM11] BPM Offensive Berlin: *BPMN 2.0 Poster*. http://www.bpmb.de/images/BPMN2_0_Poster_DE.pdf. Version: 2011

[BSB12] Businge, J. ; Serebrenik, A. ; Brand, MGJ van d.: Eclipse API usage: the good and the bad. In: *SQM, CEUR WS* (2012), S. 55–63

[BZKP09] Borovskiy, Vadym ; Zeier, Alexander ; Koch, Wolfgang ; Plattner, Hasso: Enabling enterprise composite applications on top of ERP systems. In: Kirchberg, Markus (Hrsg.) ; Hung, Patrick C. K. (Hrsg.) ;

Carminati, Barbara (Hrsg.) ; Chi, Chi-Hung (Hrsg.) ; Kanagasabai, Rajaraman (Hrsg.) ; Valle, Emanuele D. (Hrsg.) ; Lan, Kun-Chan (Hrsg.) ; Chen, Ling-Jyh (Hrsg.): *APSCC*, IEEE, 2009. – ISBN 978–1–4244–5336–8, S. 492–497

[Car03] Carnegie Mellon Software Engineering Institute: *Application Programming Interfaces.* 2003

[Car06] Carvalho, RogerioAtem: Issues on Evaluating Free/Open Source ERP Systems. Version: 2006. `http://dx.doi.org/10.1007/0-387-34456-X_72`. In: Tjoa, A.Min (Hrsg.) ; Xu, Li (Hrsg.) ; Chaudhry, SohailS. (Hrsg.): *Research and Practical Issues of Enterprise Information Systems* Bd. 205. Springer US, 2006. – DOI 10.1007/0–387–34456–X_72. – ISBN 978–0–387–34345–7, 667-675

[Cha03] Chakrabarti, S.: *Mining the Web: discovering knowledge from hypertext data.* Morgan Kaufmann Pub, 2003

[Che76] Chen, P.P.S.: The entity-relationship model?toward a unified view of data. In: *ACM Transactions on Database Systems (TODS)* 1 (1976), Nr. 1, S. 9–36

[Cla04] Clarke, Steven: *Measuring API Usability.* `http://www.drdobbs.com/windows/184405654`. Version: 2004

[Col10] Collins, M.: *Beginning WF: Windows Workflow in .NET 4.0.* Apress, 2010 (Apresspod Series). `http://books.google.de/books?id=dIPP8nYDVgIC`. – ISBN 9781430224853

[Con13] Consona Corporation: *Compiere Webseite.* `http://www.compiere.com/`. Version: 2013

[DATHKB03] Der Aalst, Wil M. ; Ter Hofstede, Arthur H. ; Kiepuszewski, Bartek ; Barros, Alistair P.: Workflow patterns. In: *Distributed and parallel databases* 14 (2003), Nr. 1, S. 5–51

[Dau10] Daughtry, J.M.: The Style and Substance of API Names. In: *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on* IEEE, 2010, S. 259–260

[Dav00] Dave Winer: *XML-RPC Specification.* `http://www.xml.com/pub/r/1199`. Version: 2000

[DFMS09] Daughtry, J.M. ; Farooq, U. ; Myers, B.A. ; Stylos, J.: API usability: report on special interest group

at CHI. In: *ACM SIGSOFT Software Engineering Notes* 34 (2009), Nr. 4, S. 27–29

[DGBP11] Dumas, Marlon ; García-Bañuelos, Luciano ; Polyvyanyy, Artem: Unraveling Unstructured Process Models. In: *Business Process Modeling Notation*. Springer, 2011, S. 1–7

[DGPS12] Doedt, Markus ; Göke, Thomas ; Pardo, Jan ; Steffen, Bernhard: Reha-Sports: The Challenge of Small Margin Healthcare Accounting. In: Margaria, Tiziana (Hrsg.) ; Steffen, Bernhard (Hrsg.): *ISoLA (2)* Bd. 7610, Springer, 2012 (Lecture Notes in Computer Science). – ISBN 978–3–642–34031–4, S. 75–77

[DH09] Dekel, U. ; Herbsleb, J.D.: Improving api documentation usability with knowledge pushing. In: *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on* IEEE, 2009, S. 320–330

[DJB+09] Doux, Guillaume ; Jouault, Frédéric ; Bézivin, Jean u. a.: Transforming BPMN process models to BPEL process definitions with ATL. In: *Fifth International Workshop on Graph-Based Tools-Grabats 2009 (co-located with TOOLS 2009)*, 2009

[Dow07] Downes, Stephen: *Why the Semantic Web Will Fail*. http://halfanhour.blogspot.de/2007/03/why-semantic-web-will-fail.html. Version: 2007

[DRRM+10] Dadam, Peter ; Reichert, Manfred ; Rinderle-Ma, Stefanie ; Lanz, Andreas ; Pryss, Rüdiger ; Predeschly, Michael ; Kolb, Jens ; Ly, Linh T. ; Jurisch, Martin ; Kreher, Ulrich ; Göser, Kevin: From ADEPT to AristaFlow BPM Suite: A Research Vision Has Become Reality. Version: 2010. http://dx.doi.org/10.1007/978-3-642-12186-9_50. In: Rinderle-Ma, Stefanie (Hrsg.) ; Sadiq, Shazia (Hrsg.) ; Leymann, Frank (Hrsg.) ; Aalst, Wil (Hrsg.) ; Mylopoulos, John (Hrsg.) ; Rosemann, Michael (Hrsg.) ; Shaw, Michael J. (Hrsg.) ; Szyperski, Clemens (Hrsg.): *Business Process Management Workshops* Bd. 43. Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–12186–9, 529-531. – 10.1007/978-3-642-12186-9_50

[Dry97] Dryer, D C.: Wizards, guides, and beyond: Rational and empirical methods for selecting optimal intelligent user interface agents. In: *Proceedings of the 2nd international*

conference on Intelligent user interfaces ACM, 1997, S. 265–268

[DS11] Doedt, M. ; Steffen, B.: Requirement-Driven Evaluation of Remote ERP-System Solutions: A Service-oriented Perspective. In: *Software Engineering Workshop (SEW), 2011 34th IEEE*, 2011. – ISSN 1550–6215, S. 57 –66

[DWC10] Dillon, T. ; Wu, C. ; Chang, E.: Cloud computing: Issues and challenges. In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on* IEEE, 2010, S. 27–33

[Erl05] Erl, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2005. – ISBN 0131858580

[Erl08] Erl, Thomas: *SOA: Entwurfsprinzipien für serviceorientierte Architektur*. Addison-Wesley, 2008

[ES85] Ericsson, K.A. ; Simon, H.A.: *Protocol analysis*. MIT press, 1985

[ESM07] Ellis, B. ; Stylos, J. ; Myers, B.: The factory pattern in api design: A usability evaluation. In: *Proceedings of the 29th international conference on Software Engineering* IEEE Computer Society, 2007, S. 302–312

[ESM10] Eisenberg, D.S. ; Stylos, J. ; Myers, B.A.: Apatite: a new interface for exploring APIs. In: *Proceedings of the 28th international conference on Human factors in computing systems* ACM, 2010, S. 1331–1334

[fab11] faberNovel: *Amazon.com - The Hidden Empire*. http://de.slideshare.net/faberNovel/amazoncom-the-hidden-empire. Version: 2011

[FFI04] Forman, Ira R. ; Forman, Nate ; Ibm, John V.: Java reflection in action. (2004)

[FK05] Fan, J. ; Kambhampati, S.: A snapshot of public web services. In: *ACM SIGMOD Record* 34 (2005), Nr. 1, S. 24–32

[Fow02] Fowler, M.: Public versus published interfaces. In: *Software, IEEE* 19 (2002), Nr. 2, S. 18 –19. http://dx.doi.org/10.1109/52.991326. – DOI 10.1109/52.991326. – ISSN 0740–7459

[FP02] Frankel, David ; Parodi, John: Using model-driven architecture to develop web services. In: *IONA Technologies white paper* (2002)

[FZ10] Farooq, U. ; Zirkler, D.: API peer reviews: a method for evaluating usability of application programming interfaces. In: *Proceedings of the 2010 ACM conference on Computer supported cooperative work* ACM, 2010, S. 207–210

[Gae07] Gaeb, Jörn: *Entwicklung eines BPEL-Plugins für das JavaABC-Framework*, Universität Dortmund, Diplomarbeit, 2007

[Gao06] Gao, Yi: BPMN-BPEL transformation and round trip engineering. In: *URL: http://www. eclarus. com/pdf/BPMN BPEL Mapping. pdf* (2006)

[Gib77] Gibson, JJ: The concept of affordances. In: *Perceiving, acting, and knowing* (1977), S. 67–82

[GJZ+11] Gerken, J. ; Jetter, H.C. ; Zöllner, M. ; Mader, M. ; Reiterer, H.: The concept maps method as a tool to evaluate the usability of APIs. In: *PART 5——-Proceedings of the 2011 annual conference on Human factors in computing systems* ACM, 2011, S. 3373–3382

[GPT12] Grill, T. ; Polacek, O. ; Tscheligi, M.: Methods towards API Usability: A Structural Analysis of Usability Problem Categories. In: *Human-Centered Software Engineering* (2012), S. 164–180

[GSSO04] Gronmo, Roy ; Skogan, David ; Solheim, Ida ; Oldevik, Jon: Model-driven web services development. In: *e-Technology, e-Commerce and e-Service, 2004. EEE'04. 2004 IEEE International Conference on* IEEE, 2004, S. 42–45

[GX09] Gong, Shuai ; Xiong, Jinhua: Interaction mismatch discovery based transformation from BPMN to BPEL. In: *Services Computing, 2009. SCC'09. IEEE International Conference on* IEEE, 2009, S. 292–299

[H+95] Hollingsworth, D. u. a.: Workflow management coalition: The workflow reference model. In: *Document Number TC00-1003* (1995), Nr. 1.1

[HAAR10] Hofstede, A. M. (Hrsg.) ; Aalst, W. M. P. d. (Hrsg.) ; Adamns, M. (Hrsg.) ; Russell, N. (Hrsg.): *Modern Business Process Automation: YAWL and its Support*

Environment. Springer, 2010 http://www.springer.com/computer+science/database+management+%26+information+retrieval/book/978-3-642-03120-5

[HAHW03] Hofstede, Arthur ter ; Aalst, Wil van d. ; Hofstede, Arthur ter ; Weske, Mathias: Business Process Management: A Survey. Version: 2003. http://dx.doi.org/10.1007/3-540-44895-0_1. In: Weske, Mathias (Hrsg.): *Business Process Management* Bd. 2678. Springer Berlin / Heidelberg, 2003. – ISBN 978–3–540–40318–0, 1019-1019. – 10.1007/3-540-44895-0_1

[Hai07] Haidasch, Robert: Get ready for the next generation of SAP business applications based on the Enterprise Service-Oriented Architecture (Enterprise SOA). In: *SAP Professional Journal* July/August (2007), S. 103–128

[Hes09] Hesseler, Martin: Customizing von ERP-Systemen. In: *Controlling & Management* 53 (2009), S. 48–55

[HL11] Hou, D. ; Li, L.: Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions. In: *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on* IEEE, 2011, S. 91–100

[Hor13] Hornung, Jens: *Model Checking im Kontext von Geschäftsprozessen*, Technische Universität Dortmund, Diplomarbeit, 2013

[HRH08] Hou, D. ; Rupakheti, C.R. ; Hoover, H.J.: Documenting and evaluating scattered concerns for framework usability: A case study. In: *Software Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific* IEEE, 2008, S. 213–220

[HS09] Hai, H. ; Sakoda, S.: SaaS and Integration best practices. In: *Fujitsu Sci. Tech. J* 45 (2009), Nr. 3, S. 257–264

[Huf09] Huff, Biran: *The Semantic Web: Impossible In Theory, Impractical in Reality*. http://bexhuff.com/2009/01/the-semantic-web-impossible-in-theory-impractical-in-reality. Version: 2009

[JEJ94] Jacobson, I. ; Ericsson, M. ; Jacobson, A.: The object advantage-business process reengineering with object technology. (1994)

[JKPM07] Jörges, Sven ; Kubczak, Christian ; Pageau, Felix ; Margaria, Tiziana: Model driven design of reliable

robot control programs using the jABC. In: *Engineering of Autonomic and Autonomous Systems, 2007. EASe'07. Fourth IEEE International Workshop on* IEEE, 2007, S. 137–148

[JMPW93] Johansson, H.J. ; Machugh, P. ; Pendlebury, A.J. ; Wheeler, W.A.: *Business process reengineering: Breakpoint strategies for market dominance*. Wiley New York, 1993

[JMS06] Jörges, Sven ; Margaria, Tiziana ; Steffen, Bernhard: FormulaBuilder: a tool for graph-based modelling and generation of formulae. In: *Proceedings of the 28th international conference on Software engineering* ACM, 2006, S. 815–818

[JMS08] Jörges, Sven ; Margaria, Tiziana ; Steffen, Bernhard: Genesys: service-oriented construction of property conform code generators. In: *Innovations in Systems and Software Engineering* 4 (2008), 361-384. http://dx.doi.org/10.1007/s11334-008-0071-2. – ISSN 1614–5046. – 10.1007/s11334-008-0071-2

[Jör13] Jörges, Sven: *Lecture Notes in Computer Science*. Bd. 7747: *Construction and Evolution of Code Generators - A Model-Driven and Service-Oriented Approach*. Springer, 2013. – ISBN 978–3–642–36126–5

[Jos08] Josuttis, N.: *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*. dpunkt-Verlag, 2008

[JXB+09] Jeong, Sae Y. ; Xie, Yingyu ; Beaton, Jack ; Myers, Brad A. ; Stylos, Jeffrey ; Ehret, Ralf ; Karstens, Jan ; Efeoglu, Arkin ; Busse, Daniela K.: Improving Documentation for eSOA APIs through User Studies. In: Pipek, Volkmar (Hrsg.) ; Rosson, Mary B. (Hrsg.) ; Ruyter, Boris E. R. (Hrsg.) ; Wulf, Volker (Hrsg.): *IS-EUD* Bd. 5435, Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–00425–4, S. 86–105

[Kar09] Karla, David: *Automatische Generierung von Softwarebausteinen zur Anbindung von SAP-Diensten an ein Business-Prozess-Management-System*, Technische Universität Dortmund, Diplomarbeit, 2009

[Kas09] Kasprzak, Piotr: *Konzeption und prototypische Umsetzung der Portierung einer EAI-Middleware auf eine agile serviceorientierte Architektur (SOA)*, Technische Universität Dortmund, Diplomarbeit, 2009

[KM12] Kiadehi, Elias F. ; Mohammadi, Shahriar: Cloud ERP: Implementation of Enterprise Resource Planning Using Cloud Computing Technology. (2012)

[KMWL09] Kopp, Oliver ; Martin, Daniel ; Wutke, Daniel ; Leymann, Frank: The difference between graph-based and block-structured business process modelling languages. In: *Enterprise Modelling and Information Systems Architecture* Citeseer, 2009

[KR11] Ko, A.J. ; Riche, Y.: The role of conceptual knowledge in API usability. In: *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on* IEEE, 2011, S. 173–176

[Kub13] Kubczak, Christian: jETI: ein serviceorientiertes framework zur high level Ausführung von Remote-Komponenten. (2013)

[KVD00] Kremers, Mark ; Van Dissel, Han: Enterprise resource planning: ERP system migrations. In: *Communications of the ACM* 43 (2000), Nr. 4, S. 53–56

[KWBE03] Kleppe, Anneke G. ; Warmer, Jos ; Bast, Wim ; Explained, MDA: *The model driven architecture: practice and promise*. 2003

[LDL03] Lindsay, A. ; Downs, D. ; Lunn, K.: Business processes?attempts to find a definition. In: *Information and Software Technology* 45 (2003), Nr. 15, S. 1015–1019

[LGZC10] Liu, F. ; Guo, W. ; Zhao, Z.Q. ; Chou, W.: SaaS integration for software cloud. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* IEEE, 2010, S. 402–409

[LKRP+08] Leitner, Florian ; Krallinger, Martin ; Rodriguez-Penagos, Carlos ; Hakenberg, Jörg ; Plake, Conrad ; Kuo, Cheng-Ju ; Hsu, Chun-Nan ; Tsai, RT ; Hung, Hsi-Chuan ; Lau, William W. u. a.: Introducing meta-services for biomedical information extraction. In: *Genome biology* 9 (2008), Nr. Suppl 2, S. S6

[LLC09] Liu, F. ; Li, L. ; Chou, W.: Communications enablement of software-as-a-service (SaaS) applications. In: *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE* IEEE, 2009, S. 1–8

[LLZ+07] Li, Y. ; Liu, Y. ; Zhang, L. ; Li, G. ; Xie, B. ; Sun, J.: An exploratory study of web services on the internet.

In: *Web Services, 2007. ICWS 2007. IEEE International Conference on* IEEE, 2007, S. 380–387

[LNMS10] LAMPRECHT, A ; NAUJOKAT, Stefan ; MARGARIA, Tiziana ; STEFFEN, Bernhard: Synthesis-based loose programming. In: *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the* IEEE, 2010, S. 262–267

[Luc12] LUCA CARACCIOLO: *Twitter verleugnet mit neuer API seine Wurzeln*. `http://t3n.de/news/twitter-verleugnet-neuer-api-413077/`. Version: 2012

[LWC07] LI, Hui ; WANG, Haiyang ; CUI, Lizhen: Automatic composition of web services based on rules and meta-services. In: *Computer Supported Cooperative Work in Design, 2007. CSCWD 2007. 11th International Conference on* IEEE, 2007, S. 496–501

[Mar11] MARGARIA, Tiziana: Formal methods in the era of service-oriented design. In: *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual* IEEE, 2011, S. 452–453

[MBD+11] *Kapitel* Customer-Oriented Business Process Management: Vision and Obstacles. In: MARGARIA, Tiziana ; BOSSELMANN, Steve ; DOEDT, Markus ; FLOYD, Barry D. ; STEFFEN, Bernhard: *In Conquering Complexity.* Springer, 2011. – ISBN 978–1–4471–2296–8

[MFS11] MARGARIA, Tiziana ; FLOYD, Barry D. ; STEFFEN, Bernhard: It simply works: Simplicity and embedded systems design. In: *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual* IEEE, 2011, S. 194–199

[MG11] MELL, P. ; GRANCE, T.: The NIST definition of cloud computing (draft). In: *NIST special publication* 800 (2011), S. 145

[MH11] MAZANEK, Steffen ; HANUS, Michael: Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language. In: *Journal of Visual Languages & Computing* 22 (2011), Nr. 1, S. 66–89

[MHS05] MERNIK, Marjan ; HEERING, Jan ; SLOANE, Anthony M.: When and how to develop domain-specific languages. In:

ACM Comput. Surv. 37 (2005), Dezember, Nr. 4, 316–344. http://dx.doi.org/10.1145/1118890.1118892. – DOI 10.1145/1118890.1118892. – ISSN 0360–0300

[Mic12] MICROSOFT: *Microsoft Dynamics NAV Website*. http://www.microsoft.com/en-us/dynamics/products/nav-overview.aspx. Version: 2012

[MJX+10] MYERS, Brad A. ; JEONG, Sae Y. ; XIE, Yingyu ; BEATON, Jack ; STYLOS, Jeffrey ; EHRET, Ralf ; KARSTENS, Jan ; EFEOGLU, Arkin ; BUSSE, Daniela K.: Studying the Documentation of an API for Enterprise Service-Oriented Architecture. In: *JOEUC* 22 (2010), Nr. 1, S. 23–51

[MNS05] MARGARIA, Tiziana ; NAGEL, Ralf ; STEFFEN, Bernhard: Remote Integration and Coordination of Verification Tools in JETI. In: *Engineering of Computer-Based Systems, IEEE International Conference on the* 0 (2005), S. 431–436. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/ECBS.2005.59. – DOI http://doi.ieeecomputersociety.org/10.1109/ECBS.2005.59. ISBN 0–7695–2308–0

[MPD10] MALESHKOVA, M. ; PEDRINACI, C. ; DOMINGUE, J.: Investigating Web APIs on the World Wide Web. In: *Web Services (ECOWS), 2010 IEEE 8th European Conference on* IEEE, 2010, S. 107–114

[MRTS98] MCLELLAN, S.G. ; ROESLER, A.W. ; TEMPEST, J.T. ; SPINUZZI, C.I.: Building more usable APIs. In: *Software, IEEE* 15 (1998), Nr. 3, S. 78–86

[MS04] MARGARIA, Tiziana ; STEFFEN, Bernhard: Lightweight coarse-grained coordination: a scalable system-level approach. In: *Int. J. Softw. Tools Technol. Transf.* 5 (2004), March, 107–123. http://dx.doi.org/10.1007/s10009-003-0119-4. – DOI 10.1007/s10009–003–0119–4. – ISSN 1433–2779

[MS06] MARGARIA, Tiziana ; STEFFEN, Bernhard: Service Engineering: Linking Business and IT. In: *IEEE Computer* 39 (2006), Nr. 10, S. 45–55

[MS08] MARGARIA, Tiziana ; STEFFEN, Bernhard: Agile IT: Thinking in User-Centric Models. In: MARGARIA, Tiziana (Hrsg.) ; STEFFEN, Bernhard (Hrsg.): *ISoLA* Bd. 17, Springer, 2008 (Communications in Computer and Information Science). – ISBN 978–3–540–88478–1, S. 490–502

[MS09] Margaria, Tiziana ; Steffen, Bernhard: Continuous Model-Driven Engineering. In: *Computer* 42 (2009), S. 106–109. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/MC.2009.315. – DOI http://doi.ieeecomputersociety.org/10.1109/MC.2009.315. – ISSN 0018–9162

[MS10] Margaria, Tiziana ; Steffen, Bernhard: Simplicity as a driver for agile innovation. In: *Computer* 43 (2010), Nr. 6, S. 90–92

[MS12] Merten, Maik ; Steffen, Bernhard: *Simplicity driven application development*. 2012

[MSHM11] Merten, Maik ; Steffen, Bernhard ; Howar, Falk ; Margaria, Tiziana: Next Generation LearnLib. In: Abdulla, Parosh A. (Hrsg.) ; Leino, K. Rustan M. (Hrsg.): *TACAS* Bd. 6605, Springer, 2011 (Lecture Notes in Computer Science). – ISBN 978–3–642–19834–2, S. 220–223

[MSR05] Margaria, Tiziana ; Steffen, Bernhard ; Reitenspiess, Manfred: Service-Oriented Design: The Roots. Version: 2005. http://dx.doi.org/10.1007/11596141_34. In: Benatallah, Boualem (Hrsg.) ; Casati, Fabio (Hrsg.) ; Traverso, Paolo (Hrsg.): *Service-Oriented Computing - ICSOC 2005* Bd. 3826. Springer Berlin / Heidelberg, 2005, 450-464. – 10.1007/11596141_34

[Mul13a] MuleSoft: *MuleSoft Website*. http://www.mulesoft.org/. Version: 2013

[Mul13b] MuleSoft: *SaaS Integration Survey - 2012 Trends - Volume II*. http://resources.mulesoft.com/SaaSIntegrationSurvey.html. Version: 2013

[Mus09] Musil, Clemens von: *JR - Integration von Statistikfunktionalität in eine Prozessmanagementumgebung*, Technische Universität Dortmund, Diplomarbeit, 2009

[Nag09] Nagel, R.: *Technische Herausforderungen modellgetriebener Beherrschung von Prozesslebenszyklen aus der Fachperspektive: Von der Anforderungsanalyse zur Realisierung*, PhD thesis, Dissertation, Technische Universität Dortmund, Lehrstuhl für Programmiersysteme, Dortmund, Deutschland, Diss., 2009

[Nas] Nasser, V.H.: The Central Role of APIs in Mashup and Cloud Platforms and Need for an Agile API Development Method.

[Nat03] NATIS, Y.V.: *Service-oriented architecture scenario.* 2003

[net12a] *NetSuite API documentation: SuiteTalk.* http://www.netsuite.com/portal/developers/resources/suitetalk-archives.shtml. Version: 2012

[net12b] *NetSuite Website.* http://www.netsuite.com. Version: 2012

[Nig13] NIGHTLABS CONSULTING GMBH: *JFire Webseite.* https://www.jfire.org. Version: 2013

[NM10] NASEHI, S.M. ; MAURER, F.: Unit tests as API usage examples. In: *Software Maintenance (ICSM), 2010 IEEE International Conference on* IEEE, 2010, S. 1–10

[Nun07] NUNEZ DI CROCE, M.: *Exploiting SAP Internals - A security analysis of the RFC Interface Implementation.* http://www.blackhat.com/presentations/bh-europe-07/Nunez-Di-Croce/Whitepaper/bh-eu-07-nunez_di_croce-WP-apr19.pdf. Version: 2007

[OAS06] OASIS, SOA: *Reference Model TC,"Reference model for Service Oriented Architecture 1.0.* 2006

[OAS07] OASIS: *Web Services Business Process Execution Language Version 2.0 - Specification.* http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Version: 2007

[O'C10] O'CALLAGHAM, Portia: The API Walkthrough Method - A lightweight method for getting early feedback about an API . In: *PLATEAU*, ACM, 2010

[ODBH06] OUYANG, Chun ; DUMAS, Marlon ; BREUTEL, Stephan ; HOFSTEDE, Arthur ter: Translating standard process models to BPEL. In: *Advanced Information Systems Engineering* Springer, 2006, S. 417–432

[ODHVDA08] OUYANG, Chun ; DUMAS, Marlon ; HM, Arthur ; VAN DER AALST, Wil M.: Pattern-based translation of BPMN process models to BPEL web services. In: *International Journal of Web Services Research (IJWSR)* 5 (2008), Nr. 1, S. 42–62

[ODTHA06] OUVANS, C ; DUMAS, Marlon ; TER HOFSTEDE, Arthur H. ; AALST, Wil M. d.: From BPMN process models to BPEL web services. In: *Web Services, 2006. ICWS'06. International Conference on* IEEE, 2006, S. 285–292

[OMG11] OMG: *Business Process Model and Notation (BPMN) Version 2.0*. http://www.omg.org/spec/BPMN/2.0/. Version: 2011

[OMG12] OMG: *In OMG's OCEB Certification Program, What is the Definition of Business Process?* http://www.omg.org/oceb/defbusinessprocess.htm. Version: 2012

[Ope13a] OPENBRAVO: *Openbravo Webseite*. http://www.openbravo.com/. Version: 2013

[Ope13b] OPENERP: *OpenERP Webseite*. https://www.openerp.com/de/. Version: 2013

[Ora12] ORACLE: *JavaDoc Website*. http://docs.oracle.com/javase/7/docs/technotes/guides/javadoc/. Version: 2012

[Oum12] OUMAROU, Moussa: *Cloud-Computing-unterstütztes Gesch äftsprozessmanagement am Beispiel von Amazon WS*, Technische Universität Dortmund, Bachelor-Arbeit, 2012

[Pap07] PAPAZOGLOU, M.: What?s in a Service? In: *Software Architecture* (2007), S. 11–28

[Par72] PARNAS, D.L.: On the criteria to be used in decomposing systems into modules. In: *Communications of the ACM* 15 (1972), Nr. 12, S. 1053–1058

[Par04] PARR, Terence J.: Enforcing strict model-view separation in template engines. In: *Proceedings of the 13th international conference on World Wide Web* ACM, 2004, S. 224–233

[Par12] PARDO, Jan: *Einfachheit als Prinzip: Eine Webanwendung zur Abrechnung von Rehasportmaßnahmen*, Technische Universität Dortmund, Diplomarbeit, 2012

[PC09] PARAMESWARAN, AV ; CHADDHA, A.: Cloud interoperability and standardization. In: *SETlabs briefings* 7 (2009), Nr. 7, S. 19–26

[PGBW09] POLYVYANYY, Artem ; GARCÍA-BAÑUELOS, Luciano ; WESKE, Mathias: Unveiling hidden unstructured regions in process models. In: *On the Move to Meaningful Internet Systems: OTM 2009*. Springer, 2009, S. 340–356

[PH09] PLETCHER, D.M. ; HOU, D.: BCC: Enhancing code completion for better API usability. In: *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on* IEEE, 2009, S. 393–394

[qui12] *Intuit Quickbooks Website*. http://quickbooks.intuit.com/. Version: 2012

[Rad05] RADEN, Neil: Shedding light on shadow IT: Is Excel running your business? In: *DSSResources. com* 26 (2005)

[RB95] RUMMLER, G.A. ; BRACHE, A.P.: *Improving Performance: How To Manage the White Space on the Organization Chart. The Jossey-Bass Management Series.* ERIC, 1995

[RD98] REICHERT, Manfred ; DADAM, Peter: Adept flex - Supporting Dynamic Changes of Workflows Without Losing Control. In: *Journal of Intelligent Information Systems* 10 (1998), 93-129. http://dx.doi.org/10.1023/A:1008604709862. – ISSN 0925–9902. – 10.1023/A:1008604709862

[Red13] REDHAT SOFTWARE - JBOSS: *jBPM Website*. http://www.jboss.org/jbpm. Version: 2013

[Res01] RESCORLA, E.: *SSL and TLS-Designing and Building Secure Systems*. Boston, 2001

[rfc98] *RFC 2478: The Simple and Protected GSS-API Negotiation Mechanism*. http://tools.ietf.org/html/rfc2478. Version: 1998

[rfc02] *RFC 3244: Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols*. http://tools.ietf.org/html/rfc3244. Version: 2002

[rfc06a] *RFC 4559: SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*. http://tools.ietf.org/html/rfc4559. Version: 2006

[rfc06b] *RFC 4757: The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows*. http://tools.ietf.org/html/rfc4757. Version: 2006

[RHS05] RICHTER, Jan-Peter ; HALLER, Harald ; SCHREY, Peter: *GI Informatiklexikon: Serviceorientierte Architektur*. http://www.gi.de/service/informatiklexikon/detailansicht/article/serviceorientierte-architektur.html. Version: 2005

[Riv04] RIVIERES, J. des: Eclipse APIs: Lines in the sand. In: *EclipseCon Retrieved March* 18 (2004), S. 2004

[RJ07] RATIU, D. ; JURJENS, J.: The reality of libraries. In: *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on* IEEE, 2007, S. 307–318

[RJ08] RATIU, D. ; JURJENS, J.: Evaluating the reference and representation of domain concepts in APIs. In: *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on* IEEE, 2008, S. 242–247

[RKS10] RAMKUMAR, S. ; KUMAR, S. ; SHIROOR, R.: Process centric guidance and tools for next generation Network Services API design. In: *Internet Multimedia Services Architecture and Application (IMSAA), 2010 IEEE 4th International Conference on* IEEE, 2010, S. 1–6

[RM06] RECKER, Jan C. ; MENDLING, Jan: On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In: *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium* Namur University Press, 2006, S. 521–532

[Rob09] ROBILLARD, M.P.: What makes apis hard to learn? answers from developers. In: *Software, IEEE* 26 (2009), Nr. 6, S. 27–34

[RSB05] RAFFELT, Harald ; STEFFEN, Bernhard ; BERG, Therese: Learnlib: a library for automata learning and experimentation. In: *Proceedings of the 10th international workshop on Formal methods for industrial critical systems* ACM, 2005, S. 62–71

[Sal08] SALESFORCE: *Integraing Salesforce.com Applications and SAP*. `http://www.salesforce.com/assets/pdf/misc/WP_SAPIntegration.pdf`. Version: 2008

[sal12a] *Salesforce Website*. `http://www.salesforce.com`. Version: 2012

[Sal12b] SALESFORCE: *Salesforce Web Services API Developer's Guide*. `http://www.salesforce.com/us/developer/docs/api/index.htm`. Version: 2012

[SAP05a] SAP AG: *BAPI Programming Guide*. `http://help.sap.com/erp2005_ehp_05/helpdata/EN/e0/9eb2370f9cbe68e10000009b38f8cf/frameset.htm`. Version: 2005

[SAP05b] SAP AG: *BAPI User Guide*. `http://help.sap.com/erp2005_ehp_05/helpdata/EN/7e/5e114a4a1611d1894c0000e829fbbd/frameset.htm`. Version: 2005

[SAP10] SAP AG: *Whitepaper: SOA made easy with SAP*. 2010

[SAP12a] SAP AG: *SAP-ERP Website*. http://www.sap.com/solutions/business-suite/erp/index.epx. Version: 2012

[SAP12b] SAP AG: *SAP Website*. http://www.sap.com/. Version: 2012

[SAP13a] SAP: *SAP Business ByDesign Website*. http://www54.sap.com/solutions/tech/cloud/software/business-management-bydesign/overview/index.html. Version: 2013

[SAP13b] SAP: *SAP Business One OnDemand Website*. http://www54.sap.com/solutions/sme/software/erp/small-business-management/on-demand/index.html. Version: 2013

[Saß07] Sassmannshausen, Dennis: *Konzeption und Entwicklung prozessgestützter E-Mail Verarbeitung in serviceorientierten Architekturen*, Technische Universität Dortmund, Diplomarbeit, 2007

[SB09] Souza, C.R.B. de ; Bentolila, D.L.M.: Automatic evaluation of API usability using complexity metrics and visualizations. In: *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on* IEEE, 2009, S. 299–302

[SC07] Stylos, J. ; Clarke, S.: Usability implications of requiring parameters in objects' constructors. In: *Proceedings of the 29th international conference on Software Engineering* IEEE Computer Society, 2007, S. 529–539

[Sch10] Schmidt, Rainer: Meta-Services as Third Dimension of Service-Oriented Enterprise Architecture. In: *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2010 14th IEEE International* IEEE, 2010, S. 157–164

[SG06] Schacher, M. ; Grässle, P.: *Agile Unternehmen durch Business Rules: Der Business Rules Ansatz*. Springer, 2006 (Xpert. press Series). http://books.google.de/books?id=yOfmqyq4XLUC. – ISBN 9783540256762

[SGB+08] Stylos, J. ; Graf, B. ; Busse, D.K. ; Ziegler, C. ; Ehret, R. ; Karstens, J.: A case study of API redesign for improved usability. In: *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on* IEEE, 2008, S. 189–192

[SHBL06] SHADBOLT, N. ; HALL, W. ; BERNERS-LEE, T.: The semantic web revisited. In: *Intelligent Systems, IEEE* 21 (2006), Nr. 3, S. 96–101

[Shi12] SHI, Jia: *Integrating Conventional ERP System with Cloud Services : From the Perspective of Cloud Service Type*, KTH, School of Information and Communication Technology (ICT), Diplomarbeit, 2012. – 41 S.

[SK11] SCHELLER, T. ; KUHN, E.: Measurable concepts for the usability of software components. In: *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* IEEE, 2011, S. 129–133

[SKLN09] SCHUMM, David ; KARASTOYANOVA, Dimka ; LEYMANN, Frank ; NITZSCHE, Jörg: On visualizing and modelling BPEL with BPMN. In: *Grid and Pervasive Computing Conference, 2009. GPC'09. Workshops at the* IEEE, 2009, S. 80–87

[Sko02] SKONNARD, Aaron: XML Files: Publishing and Discovering Web Services with DISCO and UDDI. In: *MSDN Magazine* 2 (2002)

[SM07] STYLOS, J. ; MYERS, B.: Mapping the Space of API Design Decisions. In: *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on* IEEE, 2007, S. 50–60

[Smi04] SMITH, Howard: *A response to BPELJ - Enough is enough in the field of BPM.* `http://www.fairdene.com/bpelj/BPELJ-Enough-Is-Enough.pdf`. Version: 2004

[SMN+06] STEFFEN, Bernhard ; MARGARIA, Tiziana ; NAGEL, Ralf ; JÖRGES, Sven ; KUBCZAK, Christian: Model-Driven Development with the jABC. In: BIN, Eyal (Hrsg.) ; ZIV, Avi (Hrsg.) ; UR, Shmuel (Hrsg.): *Haifa Verification Conference* Bd. 4383, Springer, 2006 (Lecture Notes in Computer Science). – ISBN 978–3–540–70888–9, S. 92–108

[Sol05] SOLUTIONS, GTA W.: *Vital eCommerce Statistics.* `http://www.gtawebsolutions.com/excerpo_e-commerce_stats.htm`. Version: 2005

[Spi09] SPIZTER, Dominik: *Modellbasierte Entwicklung von Code-Generatoren für stark eingeschränkte Ausführungsumgebungen am Beispiel der iPhone-Plattform*, Technische Universität Dortmund, Diplomarbeit, 2009

[SRC+04] SOUZA, C.R.B. de ; REDMILES, D. ; CHENG, L.T. ; MILLEN, D. ; PATTERSON, J.: Sometimes you need to see through walls: a field study of application programming interfaces. In: *Computer Supported Cooperative Work: Proceedings of the 2004 ACM conference on Computer supported cooperative work* Bd. 6, 2004, S. 63–71

[SS06] SCHEER, August-Wilhelm ; SCHNEIDER, Kristof: ARIS - Architecture of Integrated Information Systems. Version: 2006. `http://dx.doi.org/10.1007/3-540-26661-5_25`. In: BERNUS, Peter (Hrsg.) ; MERTINS, Kai (Hrsg.) ; SCHMIDT, Günter (Hrsg.): *Handbook on Architectures of Information Systems.* Springer Berlin Heidelberg, 2006 (International Handbooks on Information Systems). – ISBN 978–3–540–26661–7, 605-623. – 10.1007/3-540-26661-5_25

[STA05] SCHEER, August-Wilhelm ; THOMAS, Oliver ; ADAM, Otmar: Process Modeling using Event-Driven Process Chains. Version: 2005. `http://dx.doi.org/10.1002/0471741442.ch6`. In: *Process-Aware Information Systems.* John Wiley & Sons, Inc., 2005. – DOI 10.1002/0471741442.ch6. – ISBN 9780471741442, 119–145

[Sto10] STORZ, Dominic: *Intuitive und experimentelle Modellierung von Strategien am Beispiel von Robocode*, Technische Universität Dortmund, Diplomarbeit, 2010

[Syn13] SYNERPY GMBH: *Synerpy AvERP Webseite.* `http://www.synerpy.de/`. Version: 2013

[The13a] THE YAWL FOUNDATION: *YAWL Technical Manual.* `http://www.yawlfoundation.org/manuals/YAWLTechnicalManual2.1.pdf`. Version: 2013

[The13b] THE YAWL FOUNDATION: *YAWL User Manual.* `http://www.yawlfoundation.org/manuals/YAWLUserManual2.3.pdf`. Version: 2013

[The13c] THE YAWL FOUNDATION: *YAWL Webseite.* `http://yawlfoundation.org/`. Version: 2013

[Twi13] TWITTER: *Twitter Website.* `http://www.twitter.com/`. Version: 2013

[VDKV00] VAN DEURSEN, A. ; KLINT, P. ; VISSER, J.: Domain-specific languages: An annotated bibliography. In: *ACM Sigplan Notices* 35 (2000), Nr. 6, S. 26–36

[VR03] Van Rees, Reinout: Clarity in the usage of the terms ontology, taxonomy and classification. In: *CIB REPORT* 284 (2003), S. 432

[W3C02] W3C: *Web Services Architecture - W3C Working Draft 14 November 2002*. http://www.w3.org/TR/2002/WD-ws-arch-20021114/. Version: 2002

[W3C04a] W3C: *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/OWL-S/. Version: 2004

[W3C04b] W3C: *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema/. Version: 2004

[W3C04c] W3C: *Resource Description Framework (RDF)*. http://www.w3.org/RDF/. Version: 2004

[W3C04d] W3C: *Web Services Architecture*. http://www.w3.org/TR/ws-arch/. Version: 2004

[W3C12] W3C: *OWL 2 Web Ontology Language - Document Overview (Second Edition)*. http://www.w3.org/TR/owl2-overview/. Version: 2012

[Wat09] Watson, R.B.: Improving software API usability through text analysis: A case study. In: *Professional Communication Conference, 2009. IPCC 2009. IEEE International* IEEE, 2009, S. 1–7

[WDGW08] Weidlich, Matthias ; Decker, Gero ; Grosskopf, Alexander ; Weske, Mathias: BPEL to BPMN: The myth of a straight-forward mapping. In: *On the Move to Meaningful Internet Systems: OTM 2008*. Springer, 2008, S. 265–282

[WfM95] WfMC: *The workflow reference model - Website*. http://www.wfmc.org/reference-model.html#workflow_reference_model. Version: 1995

[Whi05] White, Stephen: Using BPMN to model a BPEL process. In: *BPTrends* 3 (2005), Nr. 3, S. 1–18

[Win06] Winkler, Christian: *Entwicklung eines jABC-Plugins zum Design von JDBC-kompatiblen Datenbankschemata*, Technische Universität Dortmund, Diplomarbeit, 2006

[WMJ10] Wu, Y.C. ; Mar, L.W. ; Jiau, H.C.: CoDocent: Support API usage with code example and API documentation. In: *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on* IEEE, 2010, S. 135–140

[wor12] *Workday Website*. http://www.workday.com. Version: 2012

[Wor13] Workflow Patterns Initiative: *Workflow Patterns home page*. http://workflowpatterns.com/. Version: 2013

[WZD07] Weinreich, Rainer ; Ziebermayr, Thomas ; Draheim, Dirk: A versioning model for enterprise services. In: *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on* Bd. 2 IEEE, 2007, S. 570–575

[xTu13] xTuple: *xTuple Webseite*. http://www.xtuple.com/. Version: 2013

[YZZ+07] Yu, Xiaofeng ; Zhang, Yan ; Zhang, Tian ; Wang, Linzhang ; Zhao, Jianhua ; Zheng, Guoliang ; Li, Xuandong: Towards a model driven approach to automatic bpel generation. In: *Model Driven Architecture-Foundations and Applications* Springer, 2007, S. 204–218

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ABAP Advanced Business Application Programming
ACL Access Control List
AES Advanced Encryption Standard
AJAX Asynchronous JavaScript and XML
AOP Aspektorientierte Programmierung
API Application Programming Interface
ARIS Architektur Integrierter Informationssysteme
ASP Active Server Pages
ASP Application Service Provider
AZS API-Zugriffssicherheit (API-Anforderung)
B1WS Business One Web Services
BAPI Business Application Programming Interface
BOR Business Object Repository
BPEL Business Process Execution Language
BPEL4WS Business Process Execution Language for Web Services
BPELJ BPEL Java Extension
BPM Business Process Management
BPMN Business Process Model and Notation (vor Version 2.0: Business Process Modeling Notation)

BPMS	Business Process Management System (Geschäftsprozessmanagementsystem)
CIM	Computer Integrated Manufacturing
COM	Component Object Model
COTS	Commercial off the Shelf
CRM	Customer Relationship Management
CRUD	Create, Read, Update, Delete
DBMS	Datenbank Management System
DCOM	Distributed Component Object Model
DI-API	Data Integration API
DISCO	Discovery
DLL	Dynamic Link Library
DMS	Document Management System
DOM	Document Object Model
DOP	Deklaration optionaler Parameter (API-Anforderung)
DSL	Domain-specific Language
DTD	Document Type Definition
DVA	Dokumentation verfügbar per API (API-Anforderung)
EAI	Enterprise Application Integration
EAM	Einfacher Authentifizierungsmechanismus (API-Anforderung)
EC2	Elastic Compute Cloud
ECC	ERP Central Component
EDV	Eingabedatenvalidierung (API-Anforderung)
eEPK	Erweiterte Ereignisgesteuerte Prozessketten
EJB	Enterprise Java Beans
EL	Expression Language
EPK	Ereignisgesteuerte Prozessketten
ERP	Enterprise Resource Planing
ESB	Enterprise Service Bus
eSOA	Enterprise Service-oriented Architecture
FTP	File Transfer Protocol

GEAR	Game-based Easy and Reverse Model-Checking
GIS	Graphische Icons, die Service symbolisieren (API-Anforderung)
GLD	Geschwindigkeit (resultierend aus Latenz und Durchsatz) (API-Anforderung)
GSSAPI	Generic Security Service API
GUI	Graphical User Interface (Grafische Benutzeroberfläche)
HCI	Human Computer Interaction
HCM	Human Capital Management
HQL	Hibernate Query Language
HRM	Human Resource Management
HTML	Hypertext Markup Language
HSQL	Hyper SQL
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrasturcture as a Service
IBM	International Business Machines Corporation
ID	Identifikator
IDE	Integrated Development Environment
IDES	International Demonstration and Education System
IKB	Intuitive und konsistente Benennungen (API-Anforderung)
InBuS	Integration of Business Software
IO	Input/Output
IP	Internet Protocol
ISN	Internationalisierte Service-Namen (API-Anforderung)
IT	Informationstechnologie
ITSy	IT Simply works bzw. It Simply works
jABC	Java Application Building Center
Java EE	Java Enterprise Edition
Java SE	Java Standard Edition

JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML Web Services
jBPM	Java Business Process Management
JCo	Java Connector
JDBC	Java Database Connectivity
JDO	Java Data Objects
JDOM	Java DOM
jETI	Java Electronic Tool Integration
JIT	Just-in-time
JNDI	Java Naming & Directory Interface
JNI	Java Native Interface
JPG	Joint Photographic (Experts) Group
JRE	Java Runtime Environment
JRI	Java R Interface
JSP	Java Server Pages
jPDL	Java Process Definition Language
JSON	Javascript Object Notation
JSR	Java Specification Requests
jUEL	Java UEL
KES	Klare und einfache Struktur (API-Anforderung)
KVD	Korrekte und vollständige Dokumentation (API-Anforderung)
KVS	Korrekte und vollständige Service-Definitionen (API-Anforderung)
KMU	Kleine und mittlere Unternehmen
LDAP	Lightweight Directory Access Protocol
LTS	Labelled Transition System
MDA	Model-driven Architecture
MDD	Model-driven Development (Modellgetriebene Entwicklung)
MDSD	Model-driven Software Development (Modellgetriebene Softwareentwicklung)
MMI	Mensch-Maschine-Interaktion

MPC	Manufacturing Planning and Control
MRP	Material Requirements Planing
MRP II	Manufacturing Resources Planing
MVEL	MVFLEX Expression Language
NAV	Navision
NIST	National Institute of Standards and Technology
NT	New Technology (Microsoft Windows Produktreihe)
NTML	NT Lan Manager
OMG	Object Management Group
ORM	Object-relational Mapping (objekt-relationale Abbildung)
OTA	One-Thing-Approach
PaaS	Platform as a Service
PDF	Portable Document Format
PHP	PHP Hypertext Preprocessor
PNG	Portable Network Graphics
POI	Poor Obfuscation Implementation (veraltet, heute kein Akronym mehr)
POJO	Plain Old Java Object
PPS	Produktionsplanung und -steuerung
PUS	Plattformunabhängigkeit durch den Einsatz von Standards (API-Anforderung)
QBDE	Quickbooks Desktop Edition
QBFC	Quickbooks Foundation Classes
QBOE	Quickbooks Online Edition
qbXML	Quickbooks XML
RDBMS	Relationales DBMS
REST	Restful State Transfer
RFC	Remote Function Call
RFID	Radio-Frequency Identification
RMI	Remote Method Invocation
RPC	Remote Procedure Call

RSD	Registry für Service-Discovery (API-Anforderung)
SAAJ	SOAP with Attachments API for Java
SaaS	Software as a Service
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung
SCM	Supply Chain Management
SDK	Software Development Kit
SDN	SAP Developer Network
SFTP	Secure FTP
SIB	Service Indepedent Building Block
SLA	Service-Level-Agreement
SLG	Service Logic Graph
SOA	Service-orientierte Architektur
SOAP	Simple Object Access Protocol (veraltet, heute kein Akronym mehr)
SOQL	Salesforce Object Query Language
SOSL	Salesforce Object Search Language
SQL	Structured Query Language
SP	Servicepack
SPNEGO	Simple and Protected GSSAPI Negotiation Mechanism
SSD	Stabilität der Service-Definitionen (API-Anforderung)
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UCC	University Competence Center
UDDI	Universal Description, Discovery and Integration
UEL	Unified Expression Language
UID	Unique Identifier
UML	Unified Modeling Language

URL	Uniform Resource Locator
VBA	Visual Basic for Applications
VVA	Validierungsregeln verfügbar per API
VZB	Voller Zugriff auf Business-Objekte (API-Anforderung)
WCF	Windows Communication Foundation
WF	Windows Workflow Foundation
WID	Work Item Definition
WS-BPEL	Web Services - Business Process Execution Language
WSC	Web Service Connector
WSDL	Web Services Descripion Language
WSIF	Web Service Invocation Framework
WWW	World Wide Web
WVA	Wertevorschläge verfügbar per API (API-Anforderung)
XAML	Extensible Application Markup Language
XMDD	eXtreme Model Driven Design
XML	eXtensible Markup Language
XOR	eXclusive Or (exklusives Oder)
XP	eXtreme Programming
XPDL	XML Process Definition Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation
YAWL	Yet Another Workflow Language